

ТРУДЫ

**ИНСТИТУТА СИСТЕМНОГО
ПРОГРАММИРОВАНИЯ РАН**

**PROCEEDINGS OF THE INSTITUTE
FOR SYSTEM PROGRAMMING OF THE RAS**

ISSN Print 2079-8156
Том 37 Выпуск 4 Часть 2

ISSN Online 2220-6426
Volume 37 Issue 1 Part 2

Институт системного
программирования
им. В.П. Иванникова РАН

Москва, 2025

ИСП **РАН**

Труды Института системного программирования РАН Proceedings of the Institute for System Programming of the RAS

Труды ИСП РАН – это издание с двойной анонимной системой рецензирования, публикующее научные статьи, относящиеся ко всем областям системного программирования, технологий программирования и вычислительной техники. Целью издания является формирование научно-информационной среды в этих областях путем публикации высококачественных статей в открытом доступе.

Издание предназначено для исследователей, студентов и аспирантов, а также практиков. Оно охватывает широкий спектр тем, включая, в частности, следующие:

- операционные системы;
- компиляторные технологии;
- базы данных и информационные системы;
- параллельные и распределенные системы;
- автоматизированная разработка программ;
- верификация, валидация и тестирование;
- статический и динамический анализ;
- защита и обеспечение безопасности ПО;
- компьютерные алгоритмы;
- искусственный интеллект.

Журнал издается по одному тому в год, шесть выпусков в каждом томе.

Поддерживается открытый доступ к содержанию издания, обеспечивая доступность результатов исследований для общественности и поддерживая глобальный обмен знаниями.

Труды ИСП РАН реферируются и/или индексируются в:

Proceedings of ISP RAS are a double-blind peer-reviewed journal publishing scientific articles in the areas of system programming, software engineering, and computer science. The journal's goal is to develop a respected network of knowledge in the mentioned above areas by publishing high quality articles on open access. The journal is intended for researchers, students, and practitioners. It covers a wide variety of topics including (but not limited to):

- Operating Systems.
- Compiler Technology.
- Databases and Information Systems.
- Parallel and Distributed Systems.
- Software Engineering.
- Software Modeling and Design Tools.
- Verification, Validation, and Testing.
- Static and Dynamic Analysis.
- Software Safety and Security.
- Computer Algorithms.
- Artificial Intelligence.

The journal is published one volume per year, six issues in each volume.

Open access to the journal content allows to provide public access to the research results and to support global exchange of knowledge. **Proceedings of ISP RAS** is abstracted and/or indexed in:



Редколлегия

Главный редактор - [Аветисян Арутюн Ишханович](#), академик РАН, доктор физико-математических наук, профессор, ИСП РАН (Москва, Российская Федерация)

Заместитель главного редактора – [Карпов Леонид Евгеньевич](#), д.т.н., ИСП РАН (Москва, Российская Федерация)

Члены редколлегии

[Воронков Андрей Анатольевич](#), доктор физико-математических наук, профессор, Университет Манчестера (Манчестер, Великобритания)

[Вирбицкайте Ирина Бонавентуровна](#), профессор, доктор физико-математических наук, Институт систем информатики им. академика А.П. Ершова СО РАН (Новосибирск, Россия)

[Коннов Игорь Владимирович](#), кандидат физико-математических наук, Технический университет Вены (Вена, Австрия)

[Ластовенский Алексей Леонидович](#), доктор физико-математических наук, профессор, Университет Дублина (Дублин, Ирландия)

[Ломазова Ирина Александровна](#), доктор физико-математических наук, профессор, Национальный исследовательский университет «Высшая школа экономики» (Москва, Российская Федерация)

[Новиков Борис Асенович](#), доктор физико-математических наук, профессор, Санкт-Петербургский государственный университет (Санкт-Петербург, Россия)

[Петренко Александр Федорович](#), доктор наук, Исследовательский институт Монреаля (Монреаль, Канада)

[Черных Андрей](#), доктор физико-математических наук, профессор, Научно-исследовательский центр CICESE (Энсенада, Баха Калифорния, Мексика)

[Шустер Ассаф](#), доктор физико-математических наук, профессор, Технион — Израильский технологический институт Technion (Хайфа, Израиль)

Адрес: 109004, г. Москва, ул. А. Солженицына, дом 25.

Телефон: +7(495) 912-44-25

E-mail: info-isp@ispras.ru

Сайт: <http://www.ispras.ru/proceedings/>

Editorial Board

Editor-in-Chief - [Arutyun I. Avetisyan](#), Academician of RAS, Dr. Sci. (Phys.–Math.), Professor, Ivannikov Institute for System Programming of the RAS (Moscow, Russian Federation)

Deputy Editor-in-Chief – [Leonid E. Karpov](#), Dr. Sci. (Eng.), Ivannikov Institute for System Programming of the RAS (Moscow, Russian Federation)

Editorial Members

[Igor Konnov](#), PhD (Phys.–Math.), Vienna University of Technology (Vienna, Austria)

[Alexey Lastovetsky](#), Dr. Sci. (Phys.–Math.), Professor, UCD School of Computer Science and Informatics (Dublin, Ireland)

[Irina A. Lomazova](#), Dr. Sci. (Phys.–Math.), Professor, National Research University Higher School of Economics (Moscow, Russian Federation)

[Boris A. Novikov](#), Dr. Sci. (Phys.–Math.), Professor, St. Petersburg University (St. Petersburg, Russian Federation)

[Alexandre F. Petrenko](#), PhD, Computer Research Institute of Montreal (Montreal, Canada)

[Assaf Schuster](#), Ph.D., Professor, Technion - Israel Institute of Technology (Haifa, Israel)

[Andrei Tchervnykh](#), Dr. Sci., Professor, CICESE Research Centre (Ensenada, Baja California, Mexico).

[Irina B. Virbitskaite](#), Dr. Sci. (Phys.–Math.), The A.P. Ershov Institute of Informatics Systems, Siberian Branch of the RAS (Novosibirsk, Russian Federation)

[Andrey Voronkov](#), Dr. Sci. (Phys.–Math.), Professor, University of Manchester (Manchester, United Kingdom)

Address: 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Tel: +7(495) 912-44-25

E-mail: info-isp@ispras.ru

Web: <http://www.ispras.ru/en/proceedings>

С о д е р ж а н и е

Архитектура системы синхронизации времени в бортовой сети SpaceWire из OCPB с поддержкой стандарта ARINC 653. <i>Русецкий И.В., Алейник В.В., Чепцов В.Ю.</i>	7
Использование программно-определяемых счетчиков для построения модели оптимального энергопотребления графического процессора. <i>Гулин А.Н., Старолетов С.М.</i>	17
DIFFuzzer: обнаружение ошибок файловых систем с помощью дифференциального фазинга серого ящика. <i>Ковалевский, В.М., Кечин В.В., Ицыксон В.М.</i>	31
Анализ структурной сложности сетей потоков работ для моделирования асинхронного взаимодействия агентов. <i>Земляной Е.О., Нестеров Р.А.</i>	47
Проверка ослабленной ленивой бездефектности для сетей Петри с данными. <i>Суворов Н.М., Ломазова И.А.</i>	69
Выбор ракурсов с помощью обратного рендеринга геометрических моделей для реконструкции материалов. <i>Санжаров В.В., Фролов В.А., Галактионов В.А.</i>	85
Распределенная платформа для анализа масштабных данных с использованием биоподобных сенсомоторных алгоритмов. <i>Потанов Д.П., Старолетов С.М.</i>	103
Автоматическая генерация рецензий к коду: эволюция инструкций и интеллектуальная фильтрация. <i>Качанов В.В.</i>	117
Извлечение знаний в ограниченной области для примеров состязательных атак «черного ящика». <i>Лукьянов К.С., Перминов А.И., Турдаков Д.Ю., Паутов М.А.</i>	133
Исследование методов машинного обучения для выявления сетевых атак. <i>Лапина М.А., Подручный Н.В., Русанов М.А., Бабенко М.Г.</i>	147
Разработка и интеграция AI-ассистента в систему управления обучением. <i>Караваева Е.А., Василевский В.И., Ланин Г.М., Прокудин Д.С.</i>	175
Интеграция подхода к визуализации данных на основе онтологии и визуализации на основе ИИ с использованием Plotly. <i>Джейранян А.Д., Лядова Л.Н.</i>	191
Разработка интеллектуальной системы на основе знаний для оценки устойчивости российских регионов. <i>Федосеев Д.С., Нерослов А.Д., Ланин В.В.</i>	209

Комбинирование логических рассуждений и LLM на пути к созданию мультиагентных систем умного дома. <i>Резуник Л.А., Прозорский М.А., Александров Д.В.</i>	221
Применение динамической символьной интерпретации в гибридном фаззинге бинарного кода для архитектур Байкал-М и RISC-V 64. <i>Логунова В.И.</i>	235
Sydr-Fuzz: непрерывный гибридный фаззинг и динамический анализ для жизненного цикла безопасной разработки. <i>Вишняков А.В., Куц Д.О., Логунова В.И., Парыгина Д.А., Кобрин И.А., Савидов Г.А., Федотов А.Н.</i>	251

Table of Contents

Architecture for time synchronization in an onboard SpaceWire network of ARINC 653 nodes.
Rusetskiy I.V., Aleinik V.V., Cheptsov V.Y...... 7

Using software-defined performance counters to construct a GPU power consumption model.
Gulin A.N., Staroletov S.M. 17

DIFFuzzer: Detecting File System Errors with Differential Grey-box Fuzzing.
Kovalevsky V.M., Kechin V.V., Itsyson V.M. 31

Evaluating Structural Complexity of Workflow Nets Modeling Asynchronous Agent Interactions.
Zemlyanoy E.O., Nesterov R.A. 47

Relaxed Lazy Soundness Verification for Data Petri nets.
Suvorov N.M., Lomazova I.A. 69

Viewpoint selection for material reconstruction using inverse rendering of geometric models.
Sanzharov V.V., Frolov V.A., Galaktionov V.A. 85

A distributed framework for large-scale data analysis using bio-inspired sensory-motor algorithms.
Potapov D.P., Staroletov S.M...... 103

Automatic Code Review Generation: Instruction Evolution and Intelligent Filtering.
Kachanov V.V...... 117

Knowledge Distillation in Local-Region for Black-Box Adversarial Examples.
Lukianov K.S., Perminov A.I., Turdakov D.Y., Pautov M.A. 133

Research of machine learning methods for detecting network attacks.
Lapina M.A., Podruchny N.V., Rusanov M.A., Babenko M.G...... 147

AI-Assistant Development and Integration into Learning Management System.
Karavaeva E.A., Vasilevskij V.I., Lanin G.M., Prokudin D.S...... 175

Integrating an Ontology-Driven Approach to Data Visualization and AI Based Visualization with Plotly.
Dzheiranian A.D., Lyadova L.N. 191

Development of Knowledge-based Intelligence for Sustainability Assessment of Russian Regions.
Fedoseev D.S., Neroslov A.D., Lanin V.V. 209

Combining Logical Reasoning and LLMs Toward Creating Multi-Agent Smart Home Systems.
Rezunik L.A., Prozorskiy M.A., Alexandrov D.V. 221

Application of Dynamic Symbolic Execution in Hybrid Fuzzing of Binary Code for
Baikal-M and RISC-V 64 Architectures.

Logunova V.I......235

Sydr-Fuzz: Continuous Hybrid Fuzzing and Dynamic Analysis for Security Development
Lifecycle.

*Vishnyakov A.V., Kuts D.O., Logunova V.I., Parygina D.A., Kobrin E.A., Savidov G.A.,
Fedotov A.N.*251

DOI: 10.15514/ISPRAS-2025-37(4)-15



Architecture for Time Synchronization in an onboard SpaceWire Network of ARINC 653 Nodes

I.V. Rusetskiy <rusetskiy@ispras.ru>

V.V. Aleinik <valeinik@ispras.ru>

V.Y. Cheptsov <cheptsov@ispras.ru>

*Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*

Abstract. In this paper we present the architecture for time synchronization in an onboard network. The architecture is specific to SpaceWire protocol and is based on mechanism of broadcast codes introduced in ECSS-E-ST-50-12C standard. We examine synchronization of real time clocks as well as synchronization of ARINC 653 node schedulers. We discuss the modification to ARINC 653 Interrupt Services required for time synchronization to operate. Achieved precision of synchronization is no worse than 5 ms.

Keywords: real-time operating system; network stack; distributed systems; ARINC 653; SpaceWire; time synchronization.

For citation: Rusetskiy I.V., Aleinik V.V., Cheptsov V.Y. Architecture for clock synchronization in an onboard SpaceWire network of ARINC 653 nodes. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025. pp. 7-16. DOI: 10.15514/ISPRAS-2025-37(4)-15.

Acknowledgments. All the members of the hard real-time operating systems development and verification group at ISP RAS for their support and ideas. Savva Mitrofanov and Rustam Subkhankulov for their hands-on contributions to the result of this paper. RC Module for their invaluable technical support regarding the internals of SpaceWire hardware.

Архитектура системы синхронизации времени в бортовой сети SpaceWire из ОСПВ с поддержкой стандарта ARINC 653

И.В. Русецкий <rusetskiy@ispras.ru>

В.В. Алейник <valeinik@ispras.ru>

В.Ю. Чепцов <cheptsov@ispras.ru>

*Институт системного программирования РАН,
Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.*

Аннотация. В данной работе предлагается архитектура системы синхронизации времени вычислителей сети. Предложенная архитектура специфична для сети SpaceWire и основывается на механизме широковещательных кодов из стандарта ECSS-E-ST-50-12C. Рассматривается не только задача синхронизации часов реального времени вычислителей, но и задача синхронизации планировщиков операционных систем стандарта ARINC 653. Обсуждаются дополнение к интерфейсу ARINC 653 Interrupt Services, необходимые для функционирования системы синхронизации времени. Достигается точность синхронизации не хуже 5 мс.

Ключевые слова: операционные системы реального времени; сетевые стеки; распределённые системы; стандартный интерфейс ARINC 653; бортовая сеть SpaceWire; синхронизация часов.

Для цитирования: И.В. Русецкий, Алейник В.В., Чепцов В.Ю. Архитектура системы синхронизации времени в бортовой сети SpaceWire из ОСПВ с поддержкой стандарта ARINC 653. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 7–16 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(4) -15.

Благодарности. Авторы благодарны всем членам группы ИСП РАН по разработке и верификации операционных систем жёсткого реального времени за их поддержку и идеи. Авторы благодарят Савву Митрофанова и Рустама Субханкулова за их практический вклад в результат этой работы, а также сотрудников НТЦ Модуль за высококвалифицированную техническую поддержку по работе с оборудованием SpaceWire.

1. Introduction

The problem of time synchronization [1] is a classical problem of distributed system theory and it has serious practical implications. Synchronicity of nodes in a network and accuracy of their time sources may affect the correctness and robustness of network interactions. Unsynchronized nodes are unable to perform data transfers to a predefined static schedule – this poses a difficulty for construction of a synchronous deterministic network and complicates organization of hot spare redundancy. At the same time, there are numerous desynchronization mechanisms: frequency drift and frequency instability of crystal oscillators, timer reset on node failure. As a result, the exact definition of «time synchronization» depends on the environment and the guarantees required for network operation.

In this work we present a scheme for time synchronization in an onboard network of nodes controlled by ARINC 653 compliant real-time operating system [2]. Orientation towards avionics and space industry has a couple of important features. First, avionic network and all of its nodes (modules) are hard real-time systems – thus, our scheme must have a relatively low overhead. Second, network use cases (synchronous timed transfers, hot spare redundancy) imply module schedule synchronization as well as astronomic time distribution. Last, spacecraft networks base on specialized physical layer protocols (SpaceWire [3], MIL-STD-1553 [4]) that have unique facilities for time distribution.

We aim for a time distribution architecture that could be adapted to any ARINC 653 compliant RTOS. Even though our scheme is based on broadcast codes special to SpaceWire, we hope it is adaptable to MIL-STD-1553 and other protocols. We propose a simplistic solution for time and schedule synchronization with precision of 5 ms.

Proposed architecture is implemented in CLOS, a real-time operating system developed at ISP RAS, and tested on hardware platforms used in space industry.

2. State of the art

2.1. Time synchronization in ARINC 653

Synchronization of time in a network of modules running an ARINC 653 compatible RTOS has a couple of distinctive features. In contrast to POSIX, ARINC 653 performs static allocation of execution time, memory and other resources to entities called «partitions». Thus, all partitions of a module execute according to a fixed schedule. Module schedule is periodic (its period is called «Major Time Frame») and consists of windows assigned to partitions. After module initialization its major time frame executes periodically until system restart or shutdown, or until module schedule is switched. Module schedule is enforced by partition scheduler, which switches partition windows based on system time. In case RTOS supports ARINC 653 extended services [5] there is a system call to switch module schedules.

If we want to build a deterministic network with timed data exchanges, we need to synchronize module schedules of different nodes. Thus, synchronization subsystem must perform time distribution and shift module schedule for one or both modules backward or forward based on accumulated time difference.

According to ARINC 653, module contains user and system partitions. System partition has privilege to access a wider selection of system calls and often contains file system, network stack and other peripheral drivers (ARINC 653 implies microkernel OS architecture).

ARINC 653 has several system calls related to system time – `TIMED_WAIT`, `PERIODIC_WAIT`, `GET_TIME` – and a subsystem for working with real time – Calendar Time API. Calendar Time API allows to read or write real time using `GET_CALENDAR_TIME` and `SET_CALENDAR_TIME` system calls. Setting of time is supported in two modes: one can set calendar time directly or temporarily slew calendar time to prevent backward time shifts or forward jumps.

ARINC 653 does not specify how one can set or adjust system time. A good mechanism to adjust time and frequency for the Unix kernel is described in [6] for the ticking timer: clock frequency is changed by adding or subtracting a fixed amount at each system timer interrupt for a calculated number of ticks. This scheme is not easily adaptable to tickless timer architecture, though.

2.2. The problem of time synchronization

There is a variety of clock synchronization problem formulations, and it is important to concentrate on one. First, it is important to distinguish logical clock synchronization [7], which aims to build a total order of events in the distributed system, from physical clock synchronization. Physical clock synchronization [8] may aim at agreement of clocks in the system (internal synchronization, «any two clocks deviate no more than Δ »), or at correspondence to some external precision time source (external synchronization, «any clock deviates from real time source no more than Δ »).

Synchronization problems can also be classified depending on symmetry of roles in a network (symmetric or asymmetric) or on presumed message propagation delay [9] («synchronous» for bound delay, «asynchronous» for unbound delay).

Approaches toward time synchronization may also be classified by their tolerance to faults [10].

Following [8], we divide the time synchronization scheme into three parts:

1. Clock source selection and resynchronization event detection.
2. Time distribution protocol.
3. Time correction mechanism.

According to all the mentioned criteria in this paper we analyze physical clock synchronization and

aim for clock agreement. Our scheme is asymmetric (also called «central master synchronization» [10]) and the network has bound propagation delay. We do not perform clock source selection and simply fix time source node. Our resynchronization event and time correction mechanism are trivial as well – we resynchronize periodically with fixed period and correct time by shifting time forward by a computed delta. This work is about time distribution over SpaceWire network and about OS support required for its operation.

2.3 Time distribution infrastructure

Physical clock synchronization may be achieved either with software or with hardware.

Network Time Protocol [11] is a time synchronization protocol widely used since its development in 1985. NTP builds on top of UDP/IP transport and distributes time from atomic clocks according to a hierarchical client-server model. On each synchronization step using a pair of transfers (from client to server and to back) NTP estimates current round-trip delay θ_i and clock offset δ_i between nodes. Pairs (θ_i, δ_i) are filtered over time and used to update current real-time clock frequency. It has a precision of around 100 μ s in fast LANs. Adaptation of NTP into a hard real-time operating system is problematic as NTP expects network stack to handle arrivals of UDP/IP packets instantaneously (in interrupt handler) while RTOS tend to prohibit device interrupts in favor of code execution determinism.

Another protocol is PTP [12]. Like NTP, PTP has a hierarchical time distribution. Best master clock algorithm (BMCA) determines optimal clock source for a given node and synchronizes with it. However, to reach precision of 1-10 μ s PTP requires hardware integration into the network layer protocol. Its integration into Ethernet is called Time-Sensitive Ethernet [13] and reaches precision of around 500 ns.

Synchronous Ethernet [14] is a hardware approach that achieves synchronization with precision of 100 ns by transmission of reference signal over a dedicated line and subsequent adjustment of receiver clock frequency with a PLL.

SpaceWire and MIL-STD-1553 have lower layer mechanisms for synchronization pulse distribution (broadcast codes for SpaceWire, «synchronize» mode code for MIL-STD-1553), however they require a separate software solution to transfer timestamps. It is important that SpaceWire broadcast codes (time markers and distributed interrupts) have an increased priority compared to data packets and do not need to wait in queues during switching – these properties allow to build network-wide distribution of time instead of hop-by-hop time distribution. SpaceWire broadcast code skew and jitter can be estimated as 14 (ESC + data character) and 10 (one preempted data character) bit-periods which is equivalent to 140 ns and 100 ns for one 100 Mbit/s link.

SpaceFibre network is analogous to SpaceWire network in terms of time-distribution mechanism, however its increased bandwidth allows implementations of heavy-weight custom hardware protocols analogous to PTP [15] with accuracy of 1 μ s.

3. Time synchronization architecture

3.1 Approach to time distribution

Our time distribution scheme is asymmetric. Network has a single fixed «master» node, that periodically broadcasts a SpaceWire distributed interrupt. All other nodes in a network are «slave» nodes. After the distributed interrupt a separate message containing a timestamp is sent.

One iteration of time distribution is as follows:

1. Master node broadcasts distributed interrupt to the network and saves current time T_1 .
2. Slave node handles the interrupt and saves moment of its reception T_2 .
3. Master node transmits value of T_1 in a separate data packet over the SpaceWire network.
4. Slave node estimates the clock offset between nodes: $\theta = T_2 - T_1$.

In order to critically analyze the approach let's focus on each step in details. What we seek is the delay and jitter introduced on each step of time distribution (Fig.1):

1. Let's focus on the time difference between broadcast code distribution and saving of current time T_1 . First of all, in case the system timer is implemented as ticking timer (in contrast to tickless timer) a timer interrupt may arrive. Timer interrupt time can be roughly estimated as 1000 instructions which is equivalent to $5 \mu\text{s}$ on a 200 MHz board ($T_{\text{tickint}} = 5 \mu\text{s}$). Secondly, there may be a wraparound jitter in the implementation of a system timer. Hardware platforms with 32-bit system counter have to account for wrapping to produce a 64-bit timestamp – they have a cycle in the timestamp generation code. Yet it is guaranteed to add only a single spare iteration in the worst case (20 instructions), which is bound by 100 ns on a 200 MHz microprocessor ($T_{\text{wraparound}} = 100 \text{ ns}$). Yet another possible source of time slew from the timer is the possible context switch: on some architectures system timer is read-protected for unprivileged code, so to read a timestamp it is required to switch to the kernel and back ($T_{\text{syscall}} = 5 \mu\text{s}$).
2. Time distribution over SpaceWire network has three major sources of delay or jitter: broadcast code transmission through data link layer, link failure, interrupt handling. As we've discussed earlier, broadcast code skew and jitter can be estimated as 140 ns and 100 ns for one 100 Mbit/s link ($T_{\text{bcsllew}} = 140 \text{ ns}$, $T_{\text{bcjitter}} = 100 \text{ ns}$). This number is multiplied by a number of links broadcast code traverses from master to slave (diameter of the network graph). Time to restart the link on error detection is $T_{\text{linkfail}} = 19.2 \mu\text{s}$. Precise synchronization is impossible in presence of link failures, and it is possible to drop time samples if link failure is detected. Interrupt state preservation and handler dispatch may introduce jitter related to caching: in case kernel stack got evicted from cache processor may require to load it from memory. Rough estimate for interrupt handling is $T_{\text{int}} = 2 \mu\text{s}$ on a 200 MHz board.
3. A probable source of error on this step is packet loss. In case one timestamp or one interrupt is lost in the network, time distribution mechanism will set a wrong association between the interrupt and the timestamp, which will result in totally incorrect clock offset. This may be handled by proper filtering of collected values of θ .

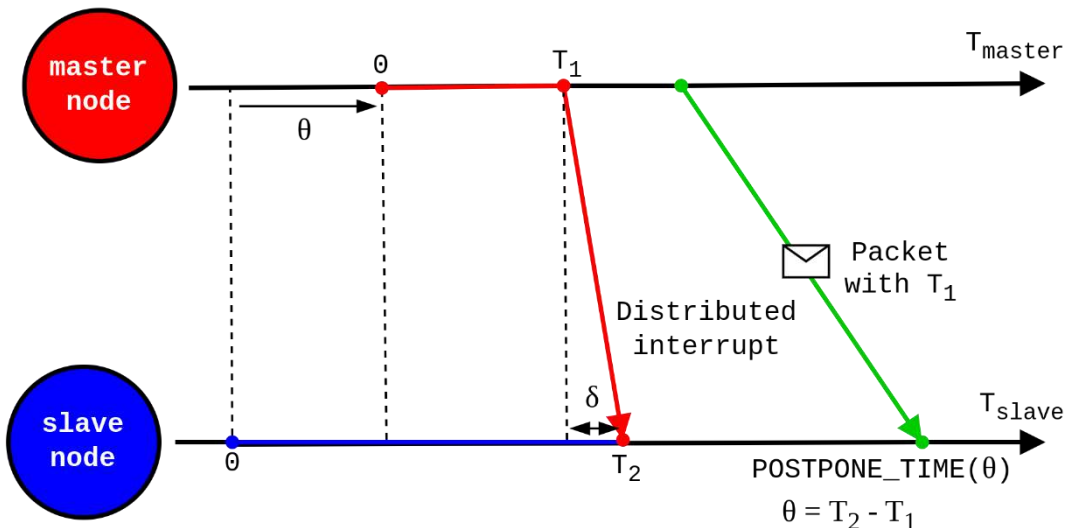


Fig. 1. One iteration of time distribution.

Our approach is simplistic and works with an ideal network that satisfies properties:

- No data packets or broadcast codes are missed or corrupted.

- Links are always functional.

A big disadvantage of our scheme is that we do not account for time propagation delay δ . At the time of writing, we do not have a methodology for precise time measurement in a distributed system, and fine-tuning is not possible at that stage.

3.2 Time synchronization subsystem In ARINC 653 RTOS

Our architecture for time synchronization subsystem in ARINC 653 real-time operating system has a set of functional software elements (Fig.2):

1. **Synchronization software.** It consists of interrupt and timestamp generation, timestamp transfer, time offset estimation and clock correction.
2. **Network stack.** It provides a simple interface for broadcast code and data packet transmission and reception, stores all hardware configurations and network routing tables.

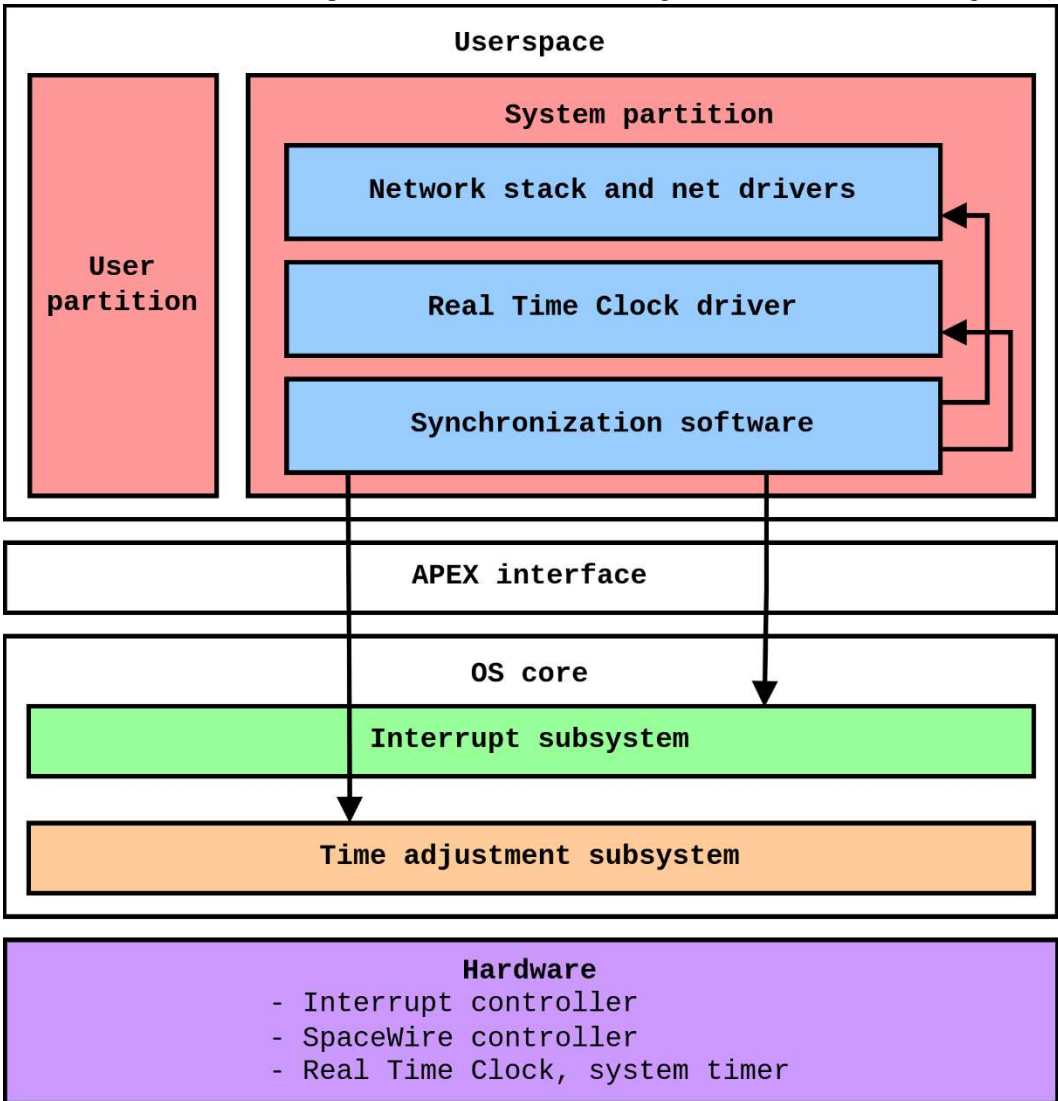


Fig. 2. Architecture for time synchronization subsystem.

3. **Interrupt handling subsystem.** Its main purpose is to perform robust partitioning of external interrupts. Only the partition that owns the interrupt source must spend time to handle the interrupt. This is achieved by unmasking the interrupt only for the duration of the interrupt source owner partition. Interrupt subsystem must provide a system call for partition to know whether a given interrupt is received.
4. **Time adjustment API.** It is important for the operating system to provide a proper set of system calls to adjust system time. The implementation of this API poses a challenge as it interferes with RTOS scheduler. Another non-standard API that turned out to be useful is the system call to get start of the current Major Time Frame and offset of the current module window.

Synchronization software, SpaceWire network stack and a real-time clock driver have to be a part of a dedicated system partition.

3.3. Synchronization software

Synchronization software is a central control unit of synchronization. Synchronization software executes periodically during every Nth Major Time Frame and contains: either master or slave time distribution algorithm, a possible role selection scheme (for now, we fix roles in the network for the sake of simplicity), a proper data filtering algorithm for received data.

For synchronization software it is important to have up-to-date status information of the corresponding SpaceWire device. For the sake of simplicity, we fix that each node in a network has only one port connected to the network: master node broadcasts distributed interrupts through a single port, slave node receives distributed interrupts from a single port.

At startup, synchronization software executes either master node algorithm or slave node algorithm (Fig. 3). Master algorithm broadcasts a distributed interrupt, saves a timestamp and transmits it to the broadcast SpaceWire logical address. Slave node handles the interrupt received from OS kernel via a system call, handles the timestamp received from SpaceWire network stack and performs a system call to adjust time.

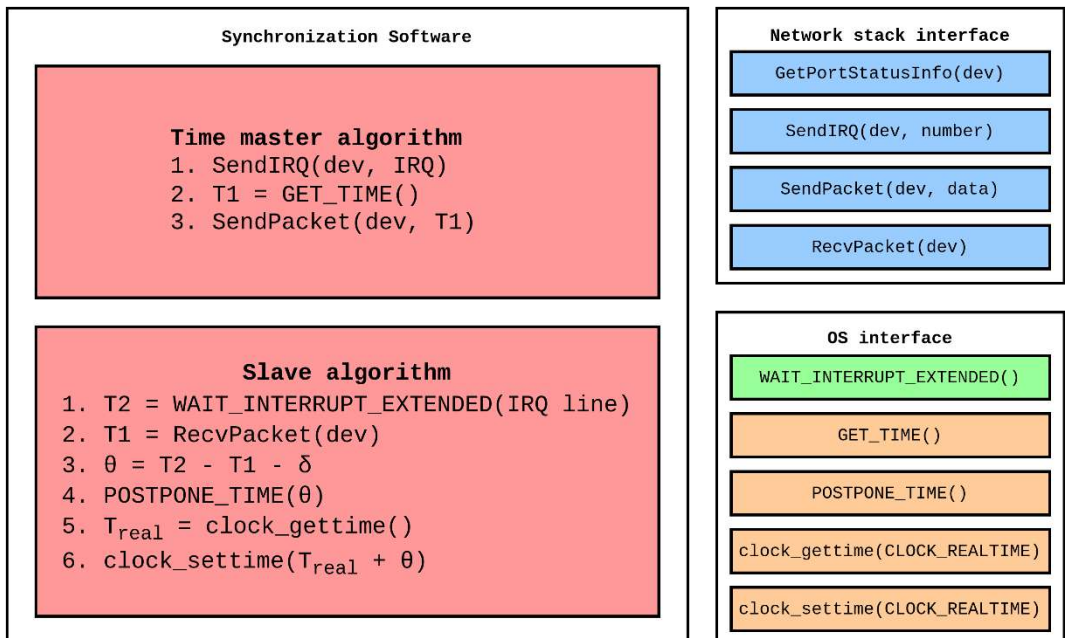


Fig. 3. Algorithms for synchronization software.

It is important to handle error scenarios:

- The clock offset value is too big (timestamp or sync pulse missed).
- The link was down since last sync pulse (timestamp or sync pulse missed, sync pulse delayed).
- The clock offset value is negative (our time adjustment only postpones time).

Described approach lacks redundancy and will not outlive permanent failure of a master node or a link to it. A possible fix to this problem: one can assign master role to three nodes in a network (each with unique interrupt id). In this scheme slave node will receive up to three distinct timestamps T_1 , T_2 , T_3 and will try to adjust its time towards median of T_1 , T_2 , T_3 .

3.4 Integration with SpaceWire Network

During the integration with SpaceWire network we faced several challenges. First challenge is the dissimilarity of time distribution interfaces provided by SpaceWire hardware. One of the IP cores we work with aimed at automatic time marker generation and the documentation did not specify how to generate a single time marker.

Second, distributed interrupts and interrupt acknowledgments are an extension to SpaceWire standard. It turned out one of the three hardware platforms we work with supports them in non-straightforward fashion.

Another problem we faced is that during link shutdown some hardware platforms do not disable interrupt generation. Thus, SpaceWire codec may recognize broadcast code symbol in the noise and generate a spurious interrupt. This could be partially mitigated by checking error bits on interrupt reception in the interrupt handling code.

3.5 Interrupt handling considerations

ARINC 653 Extended Services specifies a set of system calls that are required for a conforming OS implementation:

- `CREATE_INTERRUPT` – creates an object to be used to wait for interrupt.
- `GET_INTERRUPT_ID` – obtains ID of an interrupt configured for use by the partition.
- `WAIT_INTERRUPT` – provides a means for a process to wait upon the occurrence of a partition-specific interrupt.

None of the specified system calls transfer timestamp of synchronization pulse from the kernel, so we used `WAIT_INTERRUPT_EXTENDED` that does. Another important feature of this system call is the ability to control whether interrupt is masked during windows of other partitions. Its use enforces strict partitioning of external interrupts (which is good for system reliability and robustness) but requires nodes in a network to be already «almost synchronized» (in case an interrupt is masked and set pending during other partition window, the sync pulse will be late).

3.6 Time adjustment subsystem

We perform adjustment of both real time and system time.

Real time clock (RTC) driver is implemented in userspace and resides in system partition. However, system partition with an RTC can be a distinct partition from time synchronization partition. In this scenario client performs a remote procedure call to set time and reads a value from ARINC sampling port to get time. These two actions are wrapped in POSIX `clock_gettime/clock_settime` API familiar to most developers.

System time is a more fragile substance to work with. ARINC 653 time management services do provide a way (`GET_TIME()` APEX call) to get current time, but we need to synchronize Major

Time Frame (MTF) start times. Thus, we implemented a seqlock-based system call to observe start of the previous MTF.

In order to adjust time we used a non-standard POSTPONE_TIME() system call. Its major drawback is that it only shifts time forward (it suspends execution of current partition window and starts the next one earlier). It means that even for a little negative time offset between master node and slave node slave node will have to wait for a whole Major Time Frame.

4. Results

In order to measure accuracy of our time synchronization scheme we built a testbench of two different development boards. We marked one BSP as slave node, another one as master node, started synchronization process and observed printouts at system partition start times.

We provided output of each BSP with automatically generated timestamps and compared them. As a result of synchronization process, arbitrarily desynchronized system synchronizes to a precision of around 5 ms.

This is 3 orders of magnitude worse than what we've been expecting from estimation of distributed interrupt transfer time. However, for the moment of writing, we have a serious weakness in our measurement methodology: timestamps are generated on instrumental machine that receives output from remote synchronizing boards. Data follows a serial line at 115200 baud, OS network stack, Python multiplexing software, a series of network appliances, another OS network stack and another multiplexing software. This is hardly a suitable methodology for time measurement. A better solution would be to implement a couple of GPIO drivers, to set output pins to one on system partition start and to perform measurement with an oscilloscope.

5. Conclusion

In this paper we presented an architecture for time synchronization in an onboard SpaceWire network of ARINC 653 nodes. The developed architecture is implemented in CLOS real-time operating system, developed at ISP RAS, and tested on available hardware used in the space industry.

References

- [1]. B. Simons, J. L. Welch, and N. Lynch. 1990. An overview of clock synchronization. Fault-tolerant distributed computing. Springer-Verlag, Berlin, Heidelberg, 84–96.
- [2]. Avionics Application Software Standard Interface Part 1 – Required Services, Aeronautical Radio, Inc. ARINC Specification 653P1-5, Dec. 2019.
- [3]. SpaceWire – Links, nodes, routers, and networks, Std. ECSS-E-ST-50-12C Rev.1, May 2019.
- [4]. Review and Rationale of MIL-STD-1553A and MIL-STD-1553B, 2012.
<https://www.milstd1553.com/wp-content/uploads/2012/12/MIL-STD-1553B.pdf> (дата обращения: 14.07.2025).
- [5]. Avionics Application Software Standard Interface Part 2 – Extended Services, Aeronautical Radio, Inc. ARINC Specification 653P2-5, Dec. 2024.
- [6]. D. Mills. 1994. RFC1589: A Kernel Model for Precision Timekeeping. RFC Editor, USA.
- [7]. Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* 21, 7 (July 1978), 558–565. DOI: 10.1145/359545.359563.
- [8]. Puaud, Isabelle. A Taxonomy of Clock Synchronization Algorithms. 1997.
- [9]. B. Simons, J. L. Welch, and N. Lynch. 1990. An overview of clock synchronization. Fault-tolerant distributed computing. Springer-Verlag, Berlin, Heidelberg, 84–96.
- [10]. Kopetz, H., Steiner, W. (2022). Global Time. In: *Real-Time Systems*. Springer, Cham. DOI: 10.1007/978-3-031-11992-7_3.
- [11]. D. L. Mills, Internet time synchronization: the network time protocol, in *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482-1493, Oct. 1991, DOI: 10.1109/26.103043.

- [12]. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", in IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008), vol., no., pp.1-499, 16 June 2020, DOI: 10.1109/IEEESTD.2020.9120376.
- [13]. IEEE Standard for Local and Metropolitan Area Networks – Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks", in IEEE Std 802.1AS-2011, vol., no., pp.1-292, 30 March 2011, DOI: 10.1109/IEEESTD.2011.5741898.
- [14]. ITU-T Recommendation G.8262 (2007), Timing Characteristics of Synchronous Ethernet Equipment Slave Clock (EEC).
- [15]. E. Suvorova, Time Synchronization in SpaceFibre Networks, 2021 28th Conference of Open Innovations Association (FRUCT), Moscow, Russia, 2021, pp. 439-450, DOI: 10.23919/FRUCT50888.2021.9347624.

Информация об авторах / Information about authors

Илья Владиславович РУСЕЦКИЙ – сотрудник отдела технологий программирования ИСП РАН; магистр ВМК МГУ. Сфера научных интересов: операционные системы реального времени, распределённые системы, тестирование сетевых стеков.

Ilya Vladislavovich RUSETSKIY – employee of the department of Programming Technologies of the Ivannikov Institute for System Programming of the Russian Academy of Sciences; graduate master student at Moscow State University. Research interests: real-time operating systems, distributed systems, network stack verification.

Владислав Владимирович АЛЕЙНИК – сотрудник отдела технологий программирования ИСП РАН; аспирант МФТИ. Сфера научных интересов: операционные системы реального времени, архитектура сетевого стека, компьютерные сети SpaceWire.

Vladislav Vladimirovich ALEINIK – employee of the department of Programming Technologies of the Ivannikov Institute for System Programming of the Russian Academy of Sciences; PhD student at Moscow Institute of Physics and Technology. Research interests: real-time operating systems, network stack architecture, SpaceWire networks.

Виталий Юрьевич ЧЕПЦОВ – архитектор операционных систем в ИСП РАН; автор проекта OpenCore и регулярный контрибьютер в Tianocore EDK II и UEFITool. Занимается бортовыми операционными системами с жёстким реальным временем и исследованиями безопасности ОС общего назначения.

Vitaliy Yurievich CHEPTSOV – OS architect at ISP RAS; author of OpenCore project; contributor of Tianocore EDK II and UEFITool. Research interests: onboard hard-real time operating systems, security of general-purpose operating systems.

DOI: 10.15514/ISPRAS-2025-37(4)-16



Using Software-Defined Performance Counters to Construct a GPU Power Consumption Model

*A.N. Gulin, ORCID: 0009-0001-0519-0101 <gulin-2001@mail.ru>
S.M. Staroletov, ORCID: 0000-0001-5183-9736 <serg_soft@mail.ru>*

*Polzunov Altai State Technical University
46, prospect Lenina, Barnaul, Altai region, 656038, Russia.*

Abstract. In the modern world, processor performance and energy efficiency play a key role in computer system design. Along with CPUs, GPUs are powerful computing devices used for computer graphics processing, machine learning, and more. Processors are equipped with built-in sensors accessible through specialized tools. The chip of a modern video card can operate in a fairly wide range of frequencies and power limits (PLs). Very often, when solving a computational task or rendering a scene, the video card can operate more optimally, without wasting excess power, which can significantly save energy on labor-intensive tasks. Therefore, it is important for a set of given tasks to find such parameters where the ratio of useful work per watt will be maximum. After conducting a large number of experiments, one can learn to predict the dependence of such a target function on the parameters. This paper examines obtaining current GPU parameter values using various tools. We present results of collecting raw data from NVIDIA GPUs and the subsequent construction of an optimal power consumption model.

Keywords: GPU; NVAPI; power consumption model; GPU sensors.

For citation: Gulin A.N., Staroletov S.M. Using software-defined performance counters to construct a GPU power consumption model. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025., pp. 17-30. DOI: 10.15514/ISPRAS-2025-37(4)-16.

Использование программно-определяемых счетчиков для построения модели оптимального энергопотребления графического процессора

А.Н. Гулин, ORCID: 0009-0001-0519-0101 <gulin-2001@mail.ru>

С.М. Старолетов, ORCID: 0000-0001-5183-9736 <serg_soft@mail.ru>

АлмГТУ им. И.И. Ползунова

Россия, 656038, Алтайский край, г. Барнаул, пр. Ленина, 46.

Аннотация. В современном мире производительность и энергоэффективность процессоров играют ключевую роль при разработке компьютерных систем. Наряду с CPU, GPU являются мощными вычислительными устройствами и используются для обработки компьютерной графики, машинного обучения и многого другого. Процессоры оборудованы встроенными датчиками, доступ к которым осуществляется через специальные средства. Чип современной видеокарты может работать в достаточно широком диапазоне частот и лимитов мощности (PL). Очень часто при решении вычислительной задачи или рендеринге сцены видеокарта может работать более оптимально, без затрат избыточной мощности, что позволяет значительно экономить энергию при выполнении ресурсоёмких задач. Поэтому для заданного набора задач важно найти такие параметры, при которых отношение полезной работы к затраченной мощности будет максимальным. После проведения большого количества экспериментов можно научиться прогнозировать зависимость данной целевой функции от параметров. В данной статье рассматривается получение текущих значений параметров GPU при помощи различных инструментов. Представлены результаты сбора сырых данных с видеокарт NVIDIA и последующего построения модели оптимального энергопотребления.

Ключевые слова: графический процессор; NVAPI; модель энергопотребления; датчики видеокарты.

Для цитирования: Гулин А.Н., Старолетов С.М. Использование программно-определяемых счетчиков для построения модели оптимального энергопотребления графического процессора. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 17-30 (на английском языке). DOI: 10.15514/ISPRAS-2025-37(4)- 16.

1. Introduction

GPU (graphics processing unit) is designed to accelerate graphics rendering and parallel computations. The term “GPU” was first introduced by NVIDIA in 1999 with the release of GeForce 256. This was the first graphics card marketed as a GPU since it integrated transformation, lighting, and rendering mechanisms on a single chip, while previous graphics cards used the central processor for some of these tasks. Even in the early days of graphics cards, manufacturers transitioned to digital power management [1].

GPU differs from CPU in its architecture optimized for parallel computing. Unlike CPUs with several powerful cores, GPUs consist of thousands of simpler computing units grouped into clusters. These units are organized into streaming multiprocessors, each capable of executing multiple threads simultaneously. This structure allows GPUs to efficiently process large data volumes, such as pixels in graphics or matrices in machine learning [2].

GPUs achieve high performance computing through massive parallelism called SIMT (Single Instruction, Multiple Threads). For example, in rendering, one shader (a GPU worker) can process thousands of pixels in parallel. Additionally, GPUs use warps (NVIDIA) or wavefronts (AMD) – thread groups executing one instruction synchronously, minimizing thread management overhead [2-3].

A key aspect of GPU architecture is its complex memory hierarchy. Global memory (VRAM) has high bandwidth but significant latency, so GPUs employ multiple cache levels and registers to accelerate data access. Each streaming multiprocessor (SM) has its own shared memory enabling low-latency data exchange between threads within one block. Such process can be modeled [4-5].

Modern GPUs also use unified memory technology allowing CPUs and GPUs to work with shared address space, simplifying programming [2].

By automatically adjusting voltage and frequency parameters for the graphics processor and video memory according to GPU's BIOS algorithms, the graphics card attempts to maintain maximum frequency and performance without exceeding maximum permissible power consumption, typically specified by TDP and TGP parameters. The TDP (Thermal Design Power) parameter for graphics cards indicates the maximum heat the cooling system must dissipate [6]. The TGP (Total Graphics Power) parameter specifies total power consumption of the graphics processor, PCB, and video memory, closest to actual card consumption. Due to increased power supply requirements, power consumption must be strictly limited. The Power Limit (PL) parameter prevents exceeding calculated consumption. Adjusting PL downward can balance low power consumption with acceptable temperatures and performance [7]. When modern graphics cards reach power limits, mechanisms reduce core frequency and voltage to maintain specified consumption. Proper undervolting through voltage reduction minimally impacts performance. Excessive voltage reduction may cause system instability and crashes, requiring constant GPU temperature and performance monitoring [8]. Undervolting is completely safe: at critical voltage levels the system simply reboots.

Present work aims to study methods for obtaining raw GPU sensors data for subsequent analysis, examine existing tools for collecting such data and power consumption estimation, develop an optimized GPU power consumption model, and present results from experiments on actual hardware.

The paper is structured as follows. Section 2 reviews existing work on GPU and CPU data collection and analysis. Section 3 describes various tools for collecting GPU sensor data. Section 4 presents the developed solution for GPU data collection, model construction, as well as experimental results. The Conclusion summarizes the findings.

The present paper is an extension of the report presented at SYRCoSE Software Engineering Colloquium 2025 in Pyatigorsk.

2. Related works

This section reviews works related to obtaining and estimating power consumption for both GPUs and CPUs.

PowerAPI, described in [9], is a software package providing CPU power consumption information based on sensor data. The system comprises multiple components running as Docker containers or standalone applications (PowerAPI components can run via Python). For CPU data collection, it uses HWPC – a hardware performance counter utilizing RAPL technology for CPU monitoring. The HWPC sensor records raw data in MongoDB. The SmartWatts "formula" application then processes this data to produce power estimates. InfluxDB, specialized for time-series data, stores values computed by SmartWatts [10]. As of 2024, PowerAPI (particularly HWPC) only supports Linux and limited CPU architectures (Intel Sandy Bridge or newer, excluding Intel Core Tiger Lake, Alder Lake, Raptor Lake; AMD Zen 1, 2, 3, 4).

PowerAPI's modeling approach has two phases: training and deployment. The training phase analyzes power consumption and runs target processor events to identify those most impacting consumption. These key events form an energy model – a formula used during online deployment for real-time power estimation. The data allows PowerAPI to build a power consumption model enabling efficient real-time energy cost determination. To reduce error between predicted and actual results, models are computed for workloads with lowest average Pearson coefficient across all HWPC event combinations. Then, for each workload, the model with smallest error is selected, and from these, one model minimizing error across all workloads is chosen [11].

An Intel Xeon W3520 (Bloomfield family) running Ubuntu 14.04 (kernel 3.13) was tested with standard configuration. Power consumption estimates were obtained for parallel processes running

on the same CPU, specifically power distribution between idle consumption, the "freqmine" benchmark from PARSEC, and two other NPB benchmarks (bt.C and cg.C). Compared to physical PowerSpy measurements at 4 Hz (one power estimate per 250 ms), PowerAPI achieved 2% relative error (2.92 W) [12].

PowerAPI limitations include no GPU power modeling support, Linux-only operation, and limited CPU architecture support. Another drawback is system complexity comprising multiple separate components (requiring Docker installation).

PyJoules, described in [13], is an open-source Python toolkit (library) for measuring host machine energy consumption during Python code execution, available on GitHub and derived from "PowerAPI". PyJoules tracks energy consumed by specific host devices: socket packages for Intel CPUs, RAM (for Intel server architectures), Intel integrated graphics (client architectures), and NVIDIA GPUs. PyJoules uses Intel's "Running Average Power Limit" (RAPL) technology estimating CPU, RAM, and integrated GPU consumption, available on Intel processors since Sandy Bridge (2010).

Currently PyJoules uses Linux kernel APIs to access RAPL-reported energy values, making CPU/RAM/iGPU monitoring unavailable on Windows or MacOS. As RAPL isn't exposed in VMs, PyJoules can't monitor consumption within virtual machines.

PyJoules limitations include restricted system support (Linux only, no VMs), no AMD GPU support, no optimal power modeling capability, and inability to track overall system consumption rather than just monitored code segments. Formally, measurements are affected by all running processes, but tracking their consumption requires monitoring stub functions.

The SAOU framework in [14] combines overclocking (exceeding safe maximum frequency) and undervolting (reducing voltage below safe levels) for GPUs, achieving up to 22% energy savings without performance loss. The method uses enhanced checkpoint-recovery to correct errors from extreme settings. Testing with cuBLAS matrix multiplication showed SAOU dynamically selects optimal voltage/frequency levels based on error models. On NVIDIA GTX 980, gradual frequency increases yielded 5.3% power reduction versus baseline.

SAOU limitations include limited testing (only cuBLAS-MM), unverified effectiveness for other workloads, requiring Linux kernel modifications and GPU thread synchronization, support for GPU dynamic frequency/voltage management, additional resources for recovery systems, and prerequisite profiling for safe frequency/voltage determination.

The GreenMM framework in [15] enables energy-efficient matrix multiplication on GPUs via undervolting using algorithmic fault tolerance (ABFT). The authors experimentally determined safe undervolting boundaries and developed an error model dependent on voltage and matrix size. GreenMM combines undervolting with modified cuBLAS-MM, correcting errors dynamically. Results on NVIDIA GTX 980 show up to 19.8% power reduction and 9% energy efficiency (GFLOPS/Watt) improvement with only 1.5% performance overhead.

GreenMM limitations include limited compatibility (only via cuBLAS interface), testing only for matrix multiplication (cuBLAS-MM), requiring GPUs with NVML voltage control, no VM or non-Linux OS support, and prerequisite profiling for minimum safe voltage and error models.

Summarizing reviewed solutions, we conclude that the developed optimal power consumption system for GPU sensor data collection should differ. Existing solutions typically target Linux and matrix multiplication. As an alternative, we can develop a Windows system comprising separate modules (testing, data collection etc.), allowing test type customization based on user requests. Additionally, collected data and models from other devices with similar GPUs could be utilized. System modules could run on separate computers interacting as a distributed network.

3. Preliminaries

GPU sensor data collection can be implemented through various methods. We examine GPU-Z, NVAPI (and its PyNVML variant), and Pynvraw.

GPU-Z is free software by TechPowerUp for detailed GPU monitoring and information, Windows-only. GPU-Z provides comprehensive graphics card data (temperature, power consumption etc.) [16], supporting nearly all graphics cards.

GPU-Z's main advantage is complete GPU information collection and logging. However, closed-source code and lack of API access are drawbacks, allowing data retrieval only via processing output CSV files containing rows and columns. Reading these files makes real-time processing in custom software challenging.

NVAPI (NVIDIA Management Library) is a low-level C API provided by NVIDIA for interacting with their GPUs, Windows-only. It enables access to advanced GPU features and settings unavailable through standard drivers or APIs like OpenGL/DirectX. Unlike high-level APIs abstracting hardware details, NVAPI provides direct hardware access for fine-tuning performance and functionality [17].

NVAPI allows obtaining system information including GPU identification, driver versions, and available memory. Beyond data collection, NVAPI manages various GPU aspects like clock speeds and power management.

PyNVML is a Python wrapper for NVIDIA Management Library (NVML) providing an interface for NVIDIA GPU interaction. This library enables monitoring GPU status and characteristics like utilization, memory usage, frequency, temperature, fan speed, and more. PyNVML simplifies GPU monitoring integration into Python applications via its API for load optimization and performance improvement [18]. The library supports both sensor data collection and GPU parameter adjustment (e.g., via `nvmlDeviceSetPowerManagementLimit()`).

Pynvraw is a Python package interfacing with NVAPI for low-level NVIDIA GPU access. Pynvraw monitors and controls GPUs, accessing parameters like current temperature, clock speed, voltage, and workload [19], useful for real-time performance analysis. For our system, using Pynvraw methods (e.g., `api.get_core_voltage()`) instead of some PyNVML methods achieves higher precision comparable to GPU-Z data.

Among reviewed tools, PyNVML and Pynvraw are optimal for integration into our solution. They suit Python Windows implementation well, though limited to NVIDIA GPUs.

4. Implementation

For NVIDIA GPU optimal power consumption modeling, we propose a system of several interacting subsystems. Fig. 1 illustrates component relationships and data flow, with arrows indicating command requests or data retrieval between subsystems.

The system combines custom developments with connected systems/libraries/utilities (e.g., for GPU sensor data reading via API). Each component can be replaced thanks to interface usage (e.g., substituting MSI Kombustor with FurMark). Each system stage can run independently. The data collection and testing stage involves GPU sensor data collection, GPU testing, undervolting systems, and required connected components (GPU APIs etc.) with their interactions. Data collection and verification occur across various operation modes – when reaching extreme values (minimum consumption or maximum GPU/memory frequency) or system instability, more stable parameters are restored and the cycle continues until all parameter combinations are tested.

The Data Acquisition System handles GPU sensor data collection via low-level libraries, outputting string representations and storing in a connected database. The database can be remote depending on connection interface commands. The write method allows direct database storage without data resending. The system also verifies GPU availability and accessibility.

The GPU Testing System runs benchmarks with configurable parameters including test duration and load type, while monitoring GPU state via the Data Acquisition System. The GPU Undervolting System dynamically adjusts GPU parameters (frequencies, power) via low-level APIs, validating changes and recording results in the database. The Data Analysis System applies various methods to evaluate setting impacts on performance (FPS) and energy efficiency (FPS/W). Module

interaction uses network sockets with formatted requests including service availability checks and error handling.

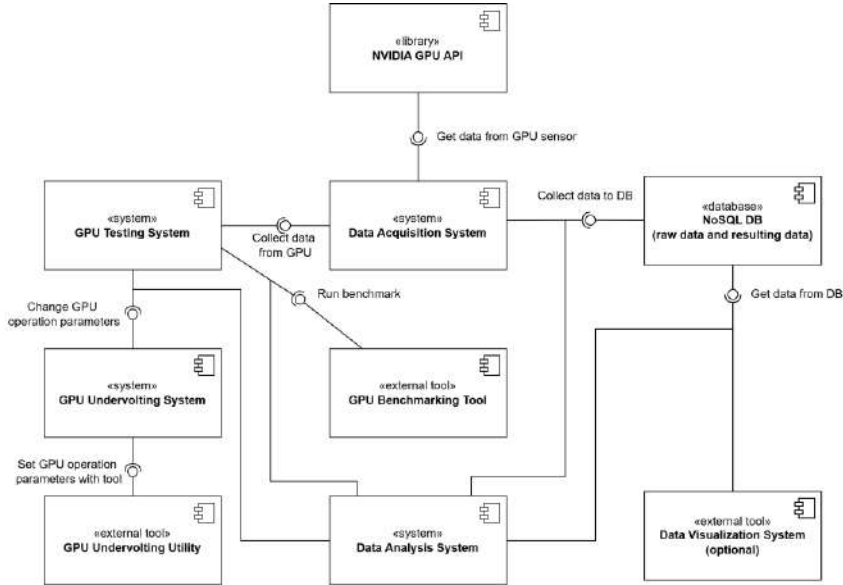


Fig. 1. Component diagram of the developed system.

The data analysis stage involves the operation of a dedicated analysis system that processes data recorded in the database during the previous stage. This system evaluates the correlation between GPU performance metrics and constructs a power consumption model to identify optimal GPU operating parameters. The model enables subsequent retesting and undervolting procedures to compare standard operating conditions with optimized configurations.

Our current Windows implementation collects NVIDIA GPU data via PyNVML and Pynvraw, adjusts GPU parameters via PyNVML and NVIDIA Inspector, runs various MSI Kombustor tests, and stores raw data in MongoDB. Below are example system execution results showing power consumption, FPS, and efficiency graphs from Metabase. "Efficiency" is a custom metric defined in (1), where F is Frames Per Second, P is Board Power Draw (W). "Efficiency" evaluates undervolting effectiveness regarding performance impact – higher values indicate better efficiency.

$$E = \frac{F}{P} \tag{1}$$

Fig. 2, 3, 4 show graphs for NVIDIA GTX 1650 data from glfurrytorus benchmark with +100 MHz GPU clock and +200 MHz memory clock offsets, reducing PL from 70W to 45W in 5W steps. Data are collected several seconds before benchmark start, during operation (FPS and efficiency values only available here), and several seconds after. This is a subset from the collected dataset – testing typically includes multiple benchmark types with varying GPU/memory clock offsets. Clock offsets increase until GPU stability is maintained, controlled via PyNVML exceptions and MSI Kombustor log checks.

As Fig. 2, 3, 4 show, reducing PL improves efficiency while lowering power consumption, but determining optimal stable values requires additional analysis.

One approach for evaluating GPU mode changes (PL and GPU/memory clock offsets) on efficiency (benchmark FPS) is computing correlation coefficients. Values near 1 indicate positive correlation between variables (FPS and GPU parameters). Pearson coefficient measures linear dependence, Kendall and Spearman coefficients measure rank correlation robust to nonlinear/asymmetric data distributions [20].

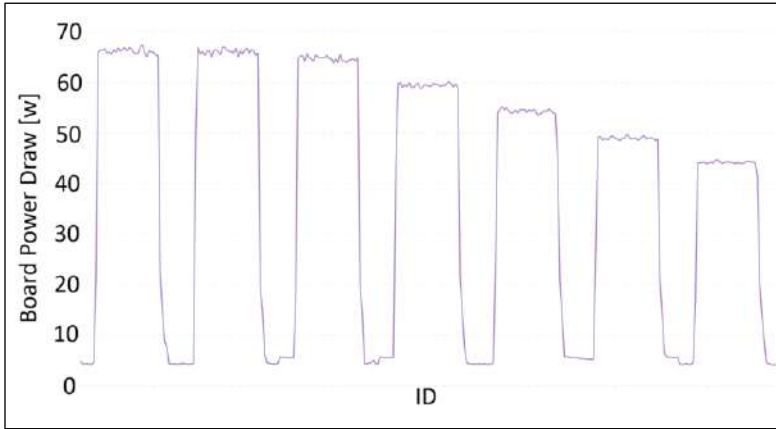


Fig. 2. Effect of Power Limit (75→45 W, 5 W Steps) on Board Power Draw (NVIDIA GTX 1650, glfurrytorus benchmark, GPU +100 MHz, memory +200 MHz).

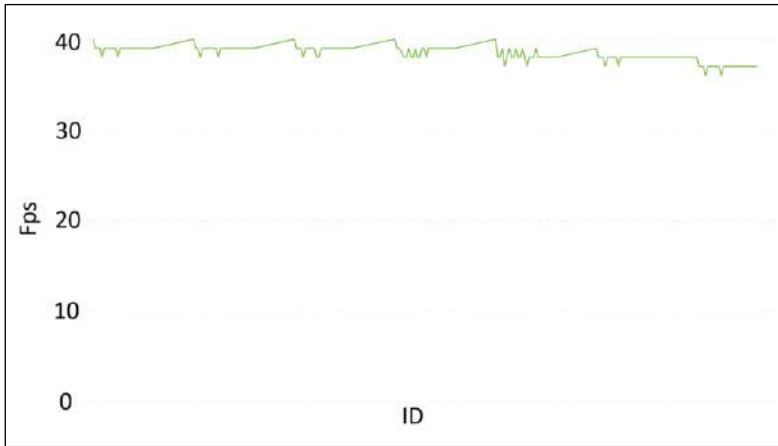


Fig. 3. Effect of Power Limit (75→45 W, 5 W Steps) on FPS (NVIDIA GTX 1650, glfurrytorus benchmark, GPU +100 MHz, memory +200 MHz).

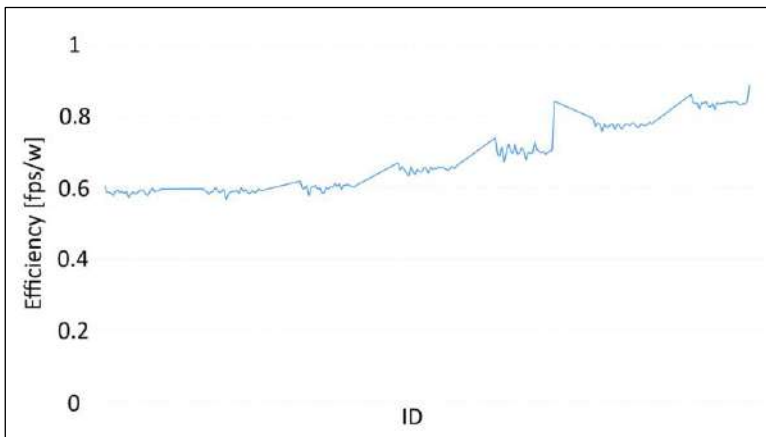


Fig. 4. Effect of Power Limit (75→45 W, 5 W Steps) on Efficiency [FPS/W] (NVIDIA GTX 1650, glfurrytorus benchmark, GPU +100 MHz, memory +200 MHz).

$$r_{xy} = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{\sqrt{[n\sum x_i^2 - (\sum x_i)^2][n\sum y_i^2 - (\sum y_i)^2]}} \quad (2)$$

$$\tau = \frac{2}{n(n-1)} \sum_{i < j} sgn(x_i - x_j) sgn(y_i - y_j) \quad (3)$$

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2 - 1)} \quad (4)$$

Pearson correlation (2) computes linear dependence between variables: x_i, y_i are compared values; n is observations count. Pearson coefficient near 1 between GPU parameter and FPS indicates strong direct linear dependence – GPU parameter increase raises FPS.

Kendall correlation (3) evaluates order consistency: sgn is the sign function (-1, 0, or 1); n is the observations count. This method counts concordant/discordant data pairs. Kendall coefficient helps precisely determine relationship direction and strength.

Spearman correlation (4) uses value ranks: d_i are rank differences; n is the observations count. Spearman coefficient between GPU parameter and FPS may reveal monotonic nonlinear relationships. Spearman helps identify stable trends between FPS and parameters when linear methods fail [21].

Table 1 shows correlation coefficients computed from 1,113,204 MongoDB documents (650,646 with FPS values). Data was collected across test types with varying PLs, GPU clock offsets, and memory clock offsets.

Table 1. Correlation coefficients for FPS performance.

Benchmark type	Power Limit [W]	GPU Clock [MHz]	Memory Clock [MHz]
Pearson Correlation			
Glfurrytorus	0.3757	0.1177	0.8230
Glfurrymsi	0.4352	0.1710	0.8095
glmsi01	0.7438	0.4665	0.0297
glmsi02gpumedium	0.2367	0.3226	0.1645
glphongdonut	0.4317	0.3927	0.5140
Glpbrdonut	0.2524	0.2171	0.7758
gtessyspherex32	0.7222	0.5988	0.0182
Kendall Correlation			
Glfurrytorus	0.2866	0.0822	0.7079
Glfurrymsi	0.3267	0.1179	0.6765
glmsi01	0.6377	0.3686	0.0228
glmsi02gpumedium	0.1717	0.2343	0.1122
glphongdonut	0.3243	0.2830	0.3833
Glpbrdonut	0.1818	0.1502	0.6058
gtessyspherex32	0.5656	0.4656	0.0129
Spearman Correlation			
Glfurrytorus	0.3636	0.1095	0.8248
Glfurrymsi	0.4253	0.1608	0.8105
glmsi01	0.7466	0.4654	0.0298
glmsi02gpumedium	0.2385	0.3315	0.1621
glphongdonut	0.4387	0.3960	0.5315
Glpbrdonut	0.2495	0.2112	0.7865
gtessyspherex32	0.7147	0.6000	0.0183

The analysis reveals that different benchmark tests exhibit distinct FPS dependence patterns on various GPU parameters. The `glfurrytorus` test demonstrates strong FPS correlation with memory clock offset (Pearson 0.82) with minimal other parameter influence, while `glfurrymsi` shows pronounced memory frequency dependence (Pearson 0.81) and moderate Power Limit impact. In contrast, `glmsi01` reveals maximum FPS correlation with Power Limit (Pearson 0.74) with negligible memory frequency effects, whereas `glmsi02gpumedium` displays weak correlations with all parameters (coefficients < 0.33) due to its PhysX workload characteristics. The `glphongdonut` test presents balanced parameter influences (coefficients 0.39-0.53), while `glpbrdonut` exhibits clear memory clock dependence (Pearson 0.78) with minor other factors. The `gltessyspherex32` test shows strong Power Limit (Pearson 0.72) and GPU clock (0.60) correlations with minimal memory impact. These findings clearly show that different benchmark tests require different optimal GPU parameters, and the final parameter values inevitably represent a compromise between performance requirements across various workloads. Graphical summary of the discussed tests is presented in Fig.5.



Fig. 5. Tests used for benchmarking (MSI Kombustor).

To obtain optimal GPU parameters, it is necessary to build its model. The process of constructing an optimal GPU power consumption model begins with data preprocessing. The data is converted into a DataFrame, and entries with missing FPS values are removed. Numerical features are normalized, and 15% noise is added to the target variable (FPS or another performance metric) to improve the model's generalization capability. The categorical feature "benchmark test type" is encoded using LabelEncoder, enabling the model to work correctly with this data type.

After preprocessing, the data is split into training and test sets while preserving the proportions of test types. A LightGBM gradient boosting model is configured with parameters optimized for the regression task. These parameters ensure a balance between accuracy and training speed. The trained model is evaluated using the R^2 and MAE metrics, followed by an analysis of feature importance through visualization, which helps understand the influence of various GPU parameters on performance.

The final stage involves optimizing GPU parameters (particularly focusing on power limit, GPU frequency, and memory frequency, as their values can be adjusted via the undervolting system). The identified optimal values are transformed back into the original range and saved to a file for later use. The model's results, including quality metrics and optimal parameters, are displayed for analysis and visualized in interactive charts for easy comparison of GPU performance under different settings.

Gradient boosting is a machine learning method based on ensemble learning, where several weak models (typically decision trees) are combined into one strong model [22]. LightGBM was chosen as the primary method due to its high efficiency and speed when processing large volumes of data.

The optimizer's algorithm is used to predict the target performance parameter for different GPU settings. In general, the target variable is a GPU performance metric, which could be operations per second (FLOPS or TOPS), hash rate (mining speed), etc. In this specific case, the performance metric is FPS (frames per second) in a benchmark. A set of parameters for the current trial is generated within a normalized range.

A complete set of parameters is created and fed into the LightGBM model to predict FPS for different test types, followed by averaging the values. This averaging ensures that the optimal parameters perform well under various GPU workloads. The final step of the method is computing the objective function (5).

Bayesian optimization was chosen due to its efficiency with large datasets. The algorithm optimizes parameters in a relatively small number of iterations [23]. After each iteration, a method is called to evaluate the new set of parameters.

$$f(\theta) = \frac{\overline{FPS} \cdot (1 - \alpha)}{P \cdot \alpha + \epsilon} \tag{5}$$

$$\overline{FPS} = \frac{1}{N} \sum_{i=1}^N \tag{6}$$

The objective function (5) represents the ratio of average FPS to power P, where $\alpha=0.5$ serves as a balancing coefficient between performance and power consumption. The term $\epsilon=10^{-9}$ is a small constant included to ensure numerical stability during division operations. In this formula, θ denotes the set of optimized GPU parameters, excluding the benchmark test type. The FPS values are averaged across all test types in equation (6).

The fundamental purpose of this formula is to establish an optimal compromise between performance (numerator) and energy efficiency (denominator). The numerator increases the objective function's value as average performance improves, while the denominator decreases it when power consumption rises. The coefficient α enables precise balancing between these two factors: when $\alpha=0$ the optimization considers only FPS, when $\alpha=1$ it considers only power consumption, and the value of 0.5 provides equal consideration to both parameters.

The collected raw data consisted of 1,113,204 documents in a MongoDB database, with 650,646 documents containing valid FPS values for the NVIDIA GTX 1650 GPU. This dataset enabled training a LightGBM model with high accuracy, achieving the following metrics: $R^2=0.947$ and $MAE=18.8$ FPS.

Fig. 6 presents the feature importance analysis showing the relative influence of different parameters on FPS values. The memory controller load (in %) demonstrates the strongest correlation with FPS, followed by memory clock offset and GPU power consumption. Memory utilization and GPU temperature show moderate influence, while the fixed minimum GPU clock frequency proved least significant as it remained constant during testing. These results clearly indicate that the benchmark tests placed substantial load on the memory subsystem.

The model execution and subsequent prediction yielded optimal parameter values as follows: power limit = 45 W, GPU clock offset = 193 MHz, and memory clock offset = 468 MHz. Fig. 7 presents a comparative analysis between tests conducted with these optimized parameters and default configuration tests. The default configuration (denoted in blue) represents standard settings with power limit = 75 W and zero clock offsets, while the orange markers indicate standard settings with minimum power limit = 45 W.

The experimental results demonstrate a significant performance improvement in terms of FPS metrics, showing an increase of +3.3% compared to the standard test configuration and +8.3% relative to the standard test with minimum power consumption. The power consumption reduction closely matches that of the standard test with power limit = 45 W (representing a 31% decrease),

while exhibiting only a marginal 0.1% increase in power consumption that is effectively offset by the corresponding performance enhancement.

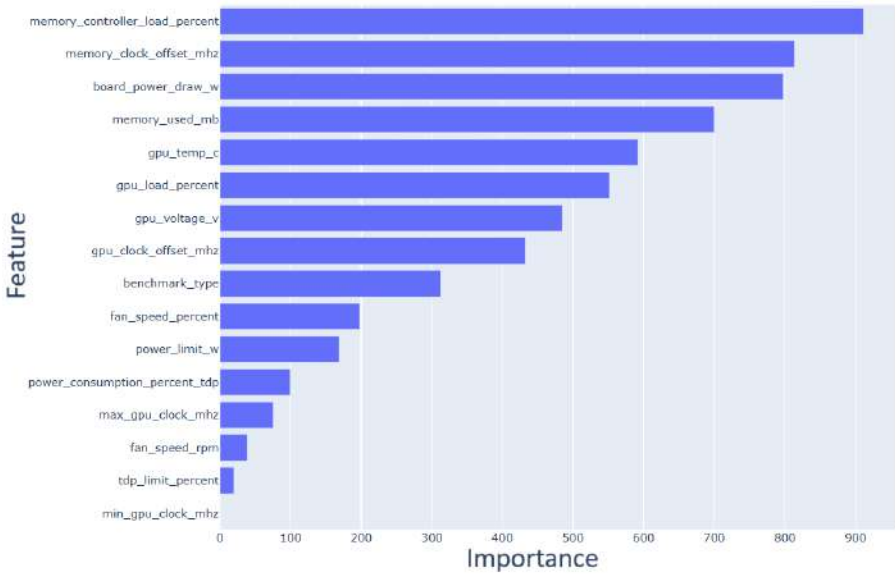


Fig. 6. Model features importance.

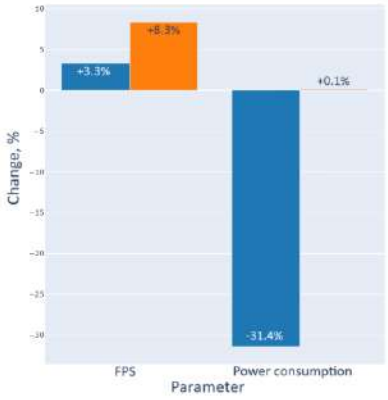


Fig. 7. Performance comparison between optimized and default parameters on GTX 1650.

5. Conclusion

Within reviewed GPU data collection tools, we examined GPU operation parameters and datasets obtainable via software-defined counters. We studied methods for obtaining graphics processor information under load. We compared existing modeling systems to determine an optimal scheme for our solution.

Current software is Python-based, we made it available in GitHub [24]. The resulting dataset, including all benchmark results and configuration parameters, is also available in [24]. We developed a modular system collecting GPU sensor data into a database, running benchmark stress test cycles, and gathering extensive information through GPU parameter adjustments via undervolting. Distributed component operation enables workload distribution across devices (e.g., data collection on one device, analysis on another). Our system performs comprehensive data

analysis to identify correlations and patterns, builds optimal power consumption models using LightGBM, and predicts optimal GPU operating parameters through Bayesian optimization methods. The obtained correlation coefficients revealed distinct optimal parameter configurations for different workload types, with final values representing a performance compromise between various workload types. The optimized parameters enabled an 8.3% FPS improvement for the NVIDIA GTX 1650 while maintaining minimal power consumption (45 W).

References

- [1]. Falk H. Personal computer graphic cards. *The Electronic Library*, vol. 12, no. 2, pp. 127–130, 1994.
- [2]. Fatahalian K., Houston M. A closer look at GPUs. *Communications of the ACM*, vol. 51, no. 10, pp. 50–57, 2008.
- [3]. Garland M, Kirk D. B. Understanding throughput-oriented architectures. *Communications of the ACM*, vol. 53, no. 11, pp. 58–66, 2010.
- [4]. Gorlatch S., Garanina N, Staroletov S. Using the SPIN model checker for auto-tuning high-performance programs. *Journal of Mathematical Sciences*, pp. 1-13, 2025.
- [5]. Кондратьев Д.А., Старолетов С.М., Шошмина И.В., Красненкова А.В., Зиборов К.В., Шилов Н.В., Гаранина Н.О., Черганов Т.Ю. Соревнования по формальной верификации VeHa-2024: накопленный в течение двух лет опыт и перспективы. *Труды ИСП РАН*, том 37, вып. 1, 2025 г., стр. 159-184. DOI: 10.15514/ISPRAS–2025–37(1)–10. / Kondratyev D.A., Staroletov S.M., Shoshmina I.V., Krasnenkova A.V., Ziborov K.V., Shilov N.V., Garanina N.O., Cherganov T.Y. VeHa-2024 Formal Verification Contest: Two Years of Experience and Prospects (in Russian). *Trudy ISP RAN/Proc. ISP RAS*, vol 37, issue. 1, pp. 159-184, 2025.
- [6]. Muralidhar R., Borovica-Gajic R, Buuya R. Energy efficient computing systems: Architectures, abstractions and modeling to techniques and standards. *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–37, 2022.
- [7]. Pei Q., Yuan Y., Hu H., Wang L., Zhang D., Yan B., Yu C., Liu F. Working Smarter Not Harder: Hybrid Cooling for Deep Learning in Edge Datacenters. *IEEE Transactions on Sustainable Computing*, 2025.
- [8]. Murdock K., Oswald D., Garcia F. D., Van Bulck J., Piessens F., Gruss D. Plundervolt: How a little bit of undervolting can create a lot of trouble. *IEEE Security & Privacy*, vol. 18, no. 5, pp. 28–37, 2020.
- [9]. Colmant M., Rouvoy R., Kurpicz M., Sobe A., Felber P., Seinturier L. The next 700 CPU power models. *Journal of Systems and Software*, vol. 144, pp. 382–396, 2018.
- [10]. Fieni G., Acero D.R., Rust P., Rouvoy R. PowerAPI: A Python framework for building software-defined power meters. *Journal of Open Source Software*, vol. 9, no. 98, p. 6670, 2024.
- [11]. Savikov D., Shintyapin I., Staroletov S. Concept of building power models for central processors using the PowerAPI toolkit (in Russian). In *Modern digital technologies: proceedings of the III All-Russian scientific-practical conference (03 June 2024)*. Barnaul: AltSTU, pp. 211–215, 2024. Available: <https://elibrary.ru/item.asp?id=68573384>.
- [12]. Gulin A., Filimontsev I., Staroletov S. Analysis of data from central processors for building their power models using the PowerAPI toolkit (in Russian). In *Modern digital technologies: proceedings of the III All-Russian scientific-practical conference (03 June 2024)*. Barnaul: AltSTU, pp. 138–142, 2024. Available: <https://elibrary.ru/item.asp?id=68573361>
- [13]. Group S.R. et al. *PyJoules: a Python library to capture the energy consumption of code snippets*. University of Lille and Inria, 2021.
- [14]. Zamani H., Tripathy D., Bhuyan L., Chen Z. SAOU: Safe adaptive overclocking and undervolting for energy-efficient GPU computing. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 205–210, 2020.
- [15]. Zamani H., Liu Y., Tripathy D., Bhuyan L, Chen Z. GreenMM: energy efficient GPU matrix multiplication through undervolting. In *Proceedings of the ACM International Conference on Supercomputing*, pp. 308–318, 2019.
- [16]. Dong Z., Wang Z., Yuan J. Research and application of GPU pass-through based on KVM in graphics workstation. In *2018 3rd International Workshop on Materials Engineering and Computer Sciences (IWMECS 2018)*. Atlantis Press, pp. 354–357, 2018.
- [17]. Czarul P., Proficz J., Krzywaniak A. Energy-aware high-performance computing: survey of state-of-the-art tools, techniques, and environments. *Scientific Programming*, vol. 2019, no. 1, p. 8348791, 2019.

- [18]. Torres L.A., Barrios C.J., Denneulin Y. Computational resource consumption in convolutional neural network training—a focus on memory. *Supercomputing Frontiers and Innovations*, vol. 8, no. 1, pp. 45–61, 2021.
- [19]. pynvraw, 2021. Available: <https://github.com/JustAMan/pynvraw>, accessed Jul. 15, 2025.
- [20]. Shaqiri M., Пjazi T., Kamberi L., Ramani-Halili R. Differences Between The Correlation Coefficients Pearson, Kendall And Spearman. *Journal of Natural Sciences and Mathematics of UT*, vol. 8, no. 15-16, pp. 392–397, 2023.
- [21]. De Winter J.C., Gosling S.D., Potter J. Comparing the Pearson and Spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data. *Psychological methods*, vol. 21, no. 3, p. 273, 2016.
- [22]. Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., Ye Q., Liu T.-Y. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, vol. 30, 2017.
- [23]. Frazier P.I. Bayesian optimization. In *Recent advances in optimization and modeling of contemporary problems*. *Inform*s, pp. 255–278, 2018.
- [24]. GPU Power Model, 2025. Available: https://github.com/GulinAlexey/GPU_Power_Model, accessed Jul. 15, 2025.

Информация об авторах / Information about authors

Алексей Николаевич ГУЛИН – магистрант кафедры прикладной математики АлтГТУ по направлению “Программная инженерия”. Сфера научных интересов: энергоэффективные вычисления, оптимизация вычислительных систем.

Alexey Nikolaevich GULIN – master student at the department of Applied Mathematics, Software Engineering direction, AltSTU. Research interests: energy-efficient computing, computational system optimization.

Сергей Михайлович СТАРОЛЕТОВ – кандидат физико-математических наук, доцент. Сфера научных интересов: формальная верификация, model checking, киберфизические системы, операционные системы.

Sergey Mikhailovich STAROLETOV – Cand. Sci. (Phys.-Math.), associate professor. Research interests: formal verification, model checking, cyber-physical systems, operating systems.

DOI: 10.15514/ISPRAS-2025-37(4)-17



DIFFuzzer: Detecting File System Errors with Differential Grey-box Fuzzing

V.M. Kovalevsky, ORCID: 0009-0002-4501-9672 <slava0135@vk.com>

V.V. Kechin, ORCID: 0009-0006-0845-2740 <valeraghst@gmail.com>

V.M. Itsykson, ORCID: 0000-0003-0276-4517 <itsykson@yandex.ru>

ITMO University,

Kronverksky Pr. 49, bldg. A, St. Petersburg, 197101, Russia.

Abstract. File systems are a crucial component of any modern operating system, whether it is a general-purpose computing system or a specialized data storage system. The cost of a file system error is very high; consequently, there is a need for effective tools for analyzing quality and detecting errors in file systems. This paper presents the DIFFuzzer tool, which is based on fuzzing techniques using grey-box and black-box principles, and implements a differential dynamic analysis approach, where the behavior of the target file system is compared to that of another, known to be of higher quality file system, which serves as a generator of reference behavior. In comparing behaviors, both the response codes of system calls and the aggregated state of the file systems are analyzed. The toolkit also includes a reducer that minimizes the erroneous trace and generates a short fragment where the error still appears. The developed tool has been tested on several POSIX-compliant file systems and has discovered several errors even during a relatively short experiment.

Keywords: file systems; differential fuzzing; file system testing; grey-box fuzzing.

For citation: Kovalevsky V.M., Kechin V.V., Itsykson V.M. DIFFuzzer: Detecting File System Errors with Differential Grey-box Fuzzing. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025, pp. 31-46. DOI: 10.15514/ISPRAS-2025-37(4)-17.

DIFFuzzer: обнаружение ошибок файловых систем с помощью дифференциального фаззинга серого ящика

В.М. Ковалевский, ORCID: 0009-0002-4501-9672 <slava0135@vk.com>

В.В. Кечин, ORCID: 0009-0006-0845-2740 <valeraghst@gmail.com>

В.М. Ицыксон, ORCID: 0000-0003-0276-4517 <itsykson@yandex.ru>

Университет ИТМО,

Россия, 197101, Санкт-Петербург, Кронверкский пр., д. 49, лит. А.

Аннотация. Файловые системы являются важнейшим компонентом любой современной операционной системы, будь то вычислительная система общего назначения или специализированная система хранения данных. Стоимость ошибки файловой системы очень высока; следовательно, существует потребность в эффективных инструментах для анализа качества и обнаружения ошибок в файловых системах. В данной статье представлен инструмент DIFFuzzer, который основан на методах фаззинга с использованием принципов серого и черного ящиков и реализует подход дифференциального динамического анализа, при котором поведение целевой файловой системы сравнивается с поведением другой, более качественной файловой системы, которая служит генератором эталонного поведения. При сравнении поведения анализируются как возвращаемые значения системных вызовов, так и агрегированное состояние файловых систем. Инструментарий также включает в себя редуктор, который минимизирует ошибочную трассу выполнения и генерирует короткий фрагмент, в котором ошибка все еще появляется. Разработанный инструмент был протестирован на нескольких файловых системах, совместимых с POSIX, и обнаружил несколько ошибок в ходе относительно короткого эксперимента.

Ключевые слова: файловые системы; дифференциальный фаззинг; тестирование файловых систем; фаззинг серого ящика.

Для цитирования: Ковалевский В.М., Кечин В.В., Ицыксон В.М. DIFFuzzer: обнаружение ошибок файловых систем с помощью дифференциального фаззинга серого ящика. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 31–46 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(4)–17.

1. Introduction

File systems embody one of the most complex components in modern computing systems, requiring extensive testing due to their critical role in data management and storage.

These considerations further emphasize the importance of rigorous testing. Most popular file systems like Ext4 have undergone extensive testing over many years of widespread use. However, traditional quality assurance methods may be insufficient for newer or specialized file systems. This is due to the vast number of possible system states and the complexity of operation sequences. Additionally, file systems are typically implemented in low-level programming languages, where memory-related errors are particularly common. For these reasons, there is a need to develop automated tools for ensuring correctness of file systems.

In this paper, we present a file system testing tool called DIFFuzzer based on differential and mutational fuzzing. The proposed approach considers the advantages and disadvantages of related works, while emphasizing the industrial relevance of the tool. Special attention is given to developing a tool that is both practical and applicable in industrial settings.

One of the main challenges was conducting an in-depth analysis of existing tools while balancing development speed with the exploration of innovative techniques. More details on related work are presented in section 3. While many tools adopt interesting and effective strategies, most are outdated or primarily academic in nature, limiting their real-world applicability. In our work, we aim to integrate these approaches into a user-friendly and efficient tool, while accounting for the constraints typical of industrial environments, such as time limitations and specific technology requirements.

The rest of the paper is organized as follows. In section 2 we state our objectives and talk about the prerequisites in detail. Research and analysis of existing works are given in section 3. Next section 4 describes problems in file system fuzzing. For example, we will elaborate on workload generation, bug detecting and reducing, execution of potential risky operations, etc. Section 5 focuses on describing our proposed approach to file system fuzzing. In section 6 we explain the details of architecture and implementation of the tool. Results of evaluation are presented in section 7. We make some conclusions, describe disadvantages and ways of development for the presented tool in section 8. Also, we share our thoughts on future bugs detection tools.

2. Motivation

The complexity of modern software development has grown considerably in recent years. At the same time, quality assurance practices have remained largely the same and often rely on unit tests and a limited number of integration or end-to-end test cases. While this may be sufficient in some cases, more sophisticated methods are necessary in highly complex projects such as file systems development, to effectively detect bugs. However, the application of scientific approaches in industrial development is not straightforward. These methods often require specialized knowledge and controlled environments, which complicates their integration into real-world systems.

Our work focuses specifically on detecting bugs in file system implementations. A file system is low-level software that interacts with both the operating system and the user. Any mistake in implementation may be critical, because file systems are used in nearly all modern computing devices.

3. Related work

In this section we review the most notable papers and tools in the domain of file system checking. Table 1 summarizes these works using criteria.

Table 1. Comparison of file system testing tools.

Name	Linux	File System Semantics	Grey-Box	FUSE	Bug Types
Syzkaller	6.14		+		Memory, Logic
kAFL	6.5		+	+	Memory, Logic
Metis	6.6	+		+	Memory, Logic, Spec
Dogfood	5.15	+		+	Memory, Logic, Crash Consistency
CrashMonkey	5.6	+			Crash Consistency
LFuzz	6.0	+	+		Memory, Logic
KRace	5.4	+	+		Memory, Logic, Data Races
Hydra	5.0	+	+	+	Memory, Logic, Spec, Crash Consistency
DIFFuzzer	6.14	+	+	+	Memory, Logic, Spec

3.1 Kernel fuzzers

Syzkaller [1] is a coverage-guided kernel fuzzer used extensively for Linux kernel testing. The fuzzer is structure-aware and uses language for modeling the kernel's programming interfaces. However, this language does not capture detailed file system semantics which makes it less effective for fuzzing file systems, as demonstrated in the Hydra study [2]. Another issue is that it uses KCov [3] for the feedback mechanism which is not suitable for FUSE [4] file systems. Also,

Syzkaller focuses on finding non-semantic bugs such as memory errors that have a clear signal, and thus it fails to trigger deep semantic bugs in file systems.

kAFL [5] is another coverage-guided kernel fuzzer. As the name suggests, it is based on AFL [6], but it uses a special feedback mechanism based on Intel Processor Trace [7], which makes it suitable for fuzzing arbitrary binaries across different operating systems. Again, the main issue with general-purpose kernel fuzzers is that they lack a model of file system semantics and are thus significantly less effective. As with Syzkaller, it is only able to find memory and logic bugs.

3.2 Black-box file system fuzzers

Metis [8] is a tool for differential fuzzing. Authors introduce an approach based on the file system model created using the SPIN [9] model checker. Metis generates workloads via SPIN, executes them on two file systems, and compares the resulting abstract file system states.

We believe differential fuzzing is particularly well-suited to file system testing due to the stable and uniform nature of file system APIs. However, aside from our work it is the only tool known to apply this approach. Because of this black-box approach Metis is only able to find “shallow” bugs.

The authors of CrashMonkey [10] present a new approach to testing file system crash consistency by simulating power-loss crashes while the workload is being executed and checking if the file system recovers to a correct state after each crash. However, CrashMonkey uses Linux kernel version 5.6 (released in 2020), which is now outdated. This is a significant limitation, as the implementation depends on a loadable kernel module. Additionally, the workload length is limited to three operations (excluding dependencies), which prevents the generation of more complex inputs. CrashMonkey is black-box and tests are generated beforehand, but even for three operations it is not possible to generate or execute all possible workloads in a reasonable time.

Dogfood [11] is a related tool with a model-based approach for workload generation using layered file system models which fixes some of CrashMonkey limitations. However, like other tools, it is not capable of finding semantic bugs, except for crash consistency bugs, similar to CrashMonkey.

3.3 Grey-box file system fuzzers

Hydra [2] is a mutation fuzzing tool that provides a comprehensive approach for file system testing. It builds upon the fuzzing infrastructure provided by AFL. A distinguishing feature of Hydra is that input data exploration is done not just through mutation of system calls but also through direct mutation of metadata stored on disk. Hydra implements mechanisms for bug detection of various classes such as logic bugs, memory errors, crash inconsistency bugs, and specification violations.

Although Hydra is the most feature-complete file system fuzzer to date, it is difficult to use in practice due to maintainability issues. The tool relies on outdated infrastructure, including Linux kernel version 5.0 (released in 2019) and Ubuntu 16.04 (whose support ended in 2021). As a result, significant effort would be required to update Hydra to modern environments.

LFuzz [12] is another file system fuzzer that exploits file system localities, expanding on Hydra. LFuzz tracks recently accessed image locations and nearby locations to predict which locations will soon be referenced, making fuzzing more effective. Unlike Hydra, LFuzz can only find memory and logic bugs. Unfortunately, the source code is not available at the moment of writing.

Another tool, KRace [13], brings coverage-guided fuzzing to the concurrency dimension. This is the only tool that attempts to address the complex challenge of testing file systems in a parallel execution environment. However, the authors do not provide documentation or instructions for using the tool or reproducing their experiments. Several repository submodules are missing, making the tool impossible to build in its current state.

3.4 Summary

Overall, only two tools presented can find specification violation bugs, where the file system fails to adhere to agreed specifications (e.g., POSIX). One of them is black-box (Metis) and the other one has not been maintained for a long time (Hydra). While not as common as memory bugs, specification violation bugs can still cause serious system failures. Based on these observations, we decided to make our own tool called DIFFuzzer to fill the gap in finding deep semantic bugs using a differential grey-box fuzzing approach.

Another challenge is the frequent abandonment of research tools, even those that demonstrate promising results. Therefore, our work emphasizes practical applicability in industrial contexts, rather than producing one-time experimental results. Syzkaller has proven that it is possible to stay relevant in the field by providing continuous support and improvements, and we will try to do the same for DIFFuzzer.

4. Problem statement

The implementation of a practical file system fuzzing tool presents numerous conceptual and engineering challenges.

4.1 Tool usability and ease of use

Other tools created by the researchers provide novel and valuable approaches to file system testing. However, it is unrealistic to use their implementations during file system development due to the lack of documentation, complexity of configuration, and limited compatibility. In this work, we place particular emphasis on usability and maintainability to ensure the tool is suitable for practical use.

4.2 Workload generation

File system fuzzing can be done by performing some workload – a sequence of file operations like creating a new file, removing existing directory etc. However, the operations within a sequence must be carefully coordinated to maintain semantic validity. For instance, writing to a deleted file would result in an invalid sequence that produces errors, and should therefore be avoided in most test cases. Indeed, sometimes it is necessary to generate invalid operations for checking error handling, but generation of sequences still must be validated against specification. For instance, POSIX [14] describes approximately 40 operations for accessing a file system. Generating valid workloads that account for the interactions and dependencies among these operations presents a significant challenge.

4.3 Workload execution

Several key considerations must be addressed when implementing workload execution:

- The effectiveness of a fuzzer is closely related to the number of executions per second. However, this is constrained by the relatively slow performance of storage devices. One possible way to improve performance is to use temporary storage, e.g., RAM [15]. This also makes it easy to reset the file system state between executions.
- State reset between executions. It is necessary to isolate workload runs as much as possible, otherwise reproducing bugs can be impossible, due to incremental system state that is hard to restore. Reset can be done by loading a virtual machine snapshot after each run or just mounting and unmounting the file system.
- Workload execution can lead to kernel panic when fuzzing the kernel file systems. All results may be lost, and the fuzzing process will stop. Because of this, kernel fuzzers execute workloads inside virtual machines.

4.4. Unexpected behaviour detection

The simplest fuzzers detect bugs by observing whether a program crashes upon execution. More advanced fuzzers implement their own mechanisms for bug detection because program crashes are just one of the many common bug types. There is a broad spectrum of bug types [2] to consider when testing file systems:

- **Memory errors.** This includes *use-after-free*, *out-of-bounds*, *null-pointer-dereference* etc., errors commonly found in programs written in low-level programming languages with direct memory access. They can be easily detected at runtime with the help of compiler instrumentation (sanitizers).
- **Logic bugs.** Errors that are triggered by failed assertions about program invariants, for example, integer division by zero, and are easy to identify as well.
- **Specification violations.** Cases where the file system fails to adhere to agreed specifications (e.g., POSIX [14]). Violations of the specification can lead to incorrect operation of application programs and are more difficult to identify.
- **Crash inconsistencies.** Cases when the system does not recover to the correct state after a failure. A file system is crash consistent if it is always restored to the correct state after a system crash due to power loss or an operating system kernel panic. Requires special facilities to detect, thus very hard to find.
- **Race conditions.** Cases where two or more threads gain unmanaged access to a shared memory without proper write synchronization or ordering. Extremely hard to detect due to the non-deterministic nature of thread interleaving.

4.5 Execution feedback

In order for a grey-box fuzzer to determine which workloads from corpus are interesting and are worth mutating later, it needs some kind of guidance or feedback, usually, code coverage.

4.6 Workload mutation

In grey-box fuzzing mutations allow to guide fuzzer towards uncovered code. Workloads that discover new coverage should be saved to the corpus for further mutations. The main challenge is making sure that mutations do not produce invalid workloads. However, in some cases, generating invalid operations is useful for testing error-handling behavior. Nevertheless, failed operations typically do not alter the file system state. Therefore, to reduce the search space and minimize workload length, it is generally preferable to avoid generating invalid operations.

4.7 Workload reduction

Workloads that trigger bugs may include numerous unrelated operations. In practical scenarios, these workloads should be minimized to help developers localize the root cause of the bug more efficiently. However, removing or simplifying operations may alter the bug's manifestation, potentially complicating its diagnosis.

5. Approach

5.1 Overview

Fig. 1 illustrates the overall workflow of DIFFuzzer. In each fuzzing iteration DIFFuzzer loads a workload from corpus (1). The workload is then mutated (2) and encoded into a C program. The program is sent to the VM and is compiled together with Executor library (3). Compiled test cases are executed on both target and reference file systems (4). If VM panics or reboots, then the potential

bug is reported (5). Otherwise, execution traces and abstract states are sent to checkers. If inconsistencies between the file systems are detected, a bug is reported (6). Alternatively, feedback from the Executor is analyzed (7). If new code coverage is discovered, the workload is saved to corpus (8) to guide further exploration. Otherwise, workload is discarded. Reduction and replication are not performed during fuzzing but can be done afterwards.

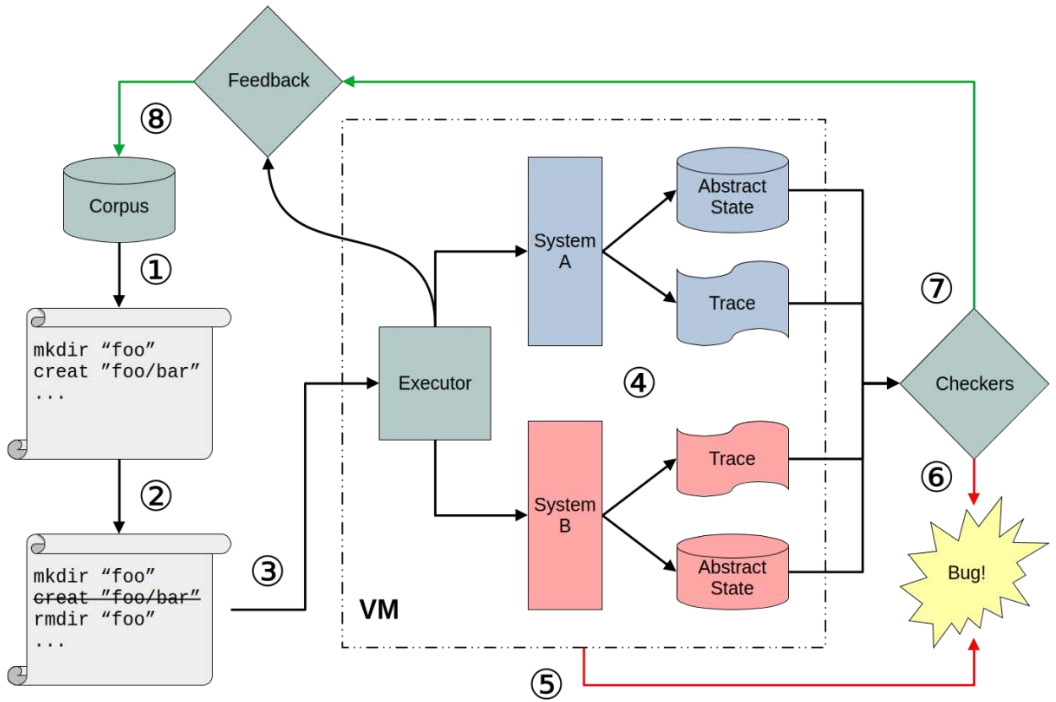


Fig. 1. Overview of DIFFuzzer workflow.

5.2 Workload generation

As was already discussed, a sequence of file operations must respect both implicit and explicit dependencies between operations. To address this, we built an abstract file system model that approximates real file system behavior. The model is capable of generating valid file operation sequences of arbitrary length. It also records all generated operations, which can later be replayed or used to generate test cases for execution on actual file systems.

The generated workloads can be exported in JSON format for internal serialization, or as C programs, which are compiled with the Executor to produce executable test binaries.

5.3 Workload execution

Most file system operations can only be invoked through the system's C API. For this reason, we developed a C++ Executor library that provides the file system API with some additional features and is used by generated C workloads.

Workloads are executed inside a virtual machine (VM), as test execution may trigger system panics or reboots – events that would be disruptive in non-isolated environments. Another reason is that using a VM allows us to set up an instrumented kernel with code coverage and sanitizers without modifying the host system. This also makes it easy to test file systems on different versions of the kernel.

5.4 Unexpected behavior detection

As was established earlier, file systems are susceptible to a wide range of bug types. Detecting memory and logic bugs is relatively straightforward, because runtime errors cause system panic (in kernel space) or program crash (in user space).

In contrast, identifying semantic bugs is significantly more challenging. Detecting specification violations requires evaluating the system's behavior against the expected specification. However, no formal specification model currently exists that can be directly utilized by automated tools. SibylFS [16] attempted to create a formal specification for file systems, but it is no longer maintained and is incompatible with modern systems.

To avoid developing a complete formal file system API specification, one can use another implementation of the same specification to detect violations using differential testing. The core idea is conceptually simple: execute identical inputs on two different file system implementations and compare their outputs. However, specifications often permit multiple valid outputs for a given input. Therefore, strict output equivalence may lead to false positives during comparison

Output comparison involves analyzing system call exit codes, error codes, and the resulting abstract file system state. The abstract state may include attributes such as file paths, sizes, link counts, permissions, and user/group identifiers – combined to form a unified representation of the file system.

Detecting crash consistency bugs and data races is significantly more difficult and remains outside the scope of this paper.

5.5 Execution feedback

Similar to many other grey-box mutational fuzzers, we use code coverage as a feedback mechanism for selecting interesting inputs. There are many ways to collect code coverage:

- **Instrumented binary.** Compiler inserts tracking code at the end of every basic block of program – linear sequence of program statements. However, kernel instrumentation cannot be performed in the same manner as for user space applications. Instead, Linux provides its own code coverage mechanism (KCOV [3]), which is used in some other kernel fuzzers like Syzkaller [1]. However, this approach has notable drawbacks: it increases binary size and RAM consumption and can reduce performance by over 100%.
- **Intel Processor Trace.** Technology available in modern Intel CPUs [7] that allows efficient tracing of all the instructions executed by a process. Its main advantages include low performance overhead (less than 5%) and applicability to closed-source targets, including operating systems. However, as demonstrated in kAFL [5], this method requires extending both QEMU [17] and KVM [18].
- **QEMU CPU emulation.** Used in TriforceAFL [19] and able to capture coverage for any binary, but with one major drawback: it can be up to 50 times slower, compared to native performance.

We chose to use instrumented binaries, as extending QEMU and KVM to support Intel Processor Trace would introduce significant engineering overhead. Additionally, QEMU-based emulation is prohibitively slow for file system fuzzing, operating at an order of magnitude below user space fuzzers. KCOV is limited to kernel-space file systems. However, we also aim to support FUSE-based [4] file systems, which can be implemented in various programming languages and may require different coverage mechanisms. Currently, we support LCOV [20] coverage for C++ and C, but we plan to add support for other coverage formats as well.

5.6 Workload mutation

Workloads are sequences of file operations. Selected workloads are mutated by inserting or removing operations at specific positions in the sequence. Mutations are implemented by replaying sequences in the abstract file system model before the point, applying the mutation at the index and replaying the second half of the workload after the point. If the replay results in an error, the mutation is discarded and a new mutation is attempted. This ensures that mutated workloads are valid, unless there is a bug in the model itself. Bugs in the model can be also found by fuzzing. If both file system traces contain errors, then it can be considered a bug in the model.

5.7 Workload reduction

Bug reduction consists of trying to remove a single operation from the end at a time. Removing operations starting from the end in succession is enough because each operation can only depend on previous operations. If the reduction can remove an operation without affecting the result, it is considered valid. This, in turn, may enable the removal of earlier operations. Sometimes, removing an operation may reveal a different bug than the original one. We suggest keeping all the bug variants. They are saved and can be checked manually later, but only the original bug is reduced further. It should be noted that if a bug is found by difference between trace exit codes, then we can trim workload up to that operation and go to the described algorithm.

6. Implementation

Implementation consists of three major parts:

- **Executor.** C++ runtime, used to execute tests.
- **Dash.** Differential abstract state hasher, used to compare file system states, written in Rust.
- **DIFFuzzer.** Fuzzer itself, also written in Rust.

We picked Rust for our tools, because fuzzing benefits from high execution speed, and Rust performance is on par with C++, which is commonly used for implementing fuzzers. However, C++ is not very ergonomic and developer experience with Rust is better, especially when using developer tools.

Another reason is that in earlier stages of development we used LibAFL [21], fuzzing framework made in Rust, in order to reuse some existing fuzzing components. Unfortunately, we quickly discovered it wasn't very suitable for our specific case.

Executor and Dash are reusable components that can operate independently and potentially can be integrated into other file system testing frameworks.

6.1 Executor

Generated workloads produce two types of files. JSON files are used for serialization and deserialization in order to save and load tests from disk. C files are used exclusively for execution and are linked against runtime to produce a test binary. These operations do not directly invoke system calls. Instead, they call a runtime library that wraps system functionality, tracks operation results (e.g., exit and error codes), and collects kernel coverage via KCov [3].

There are minor differences between real operations and those provided by the runtime. For instance, according to POSIX [14], the *creat* function should leave the file descriptor open, similar to *open* function. However, our runtime closes it immediately, as writing to the file is not always necessary. Often, file system bugs just require the file entry to exist in the directory.

The runtime is implemented in C++, unlike other components, because Linux interfaces, especially KCov functions, are not trivial to use in any other language.

At the moment Executor supports the following operations: *mkdir*, *creat*, *rmdir*, *unlink*, *symlink*, *hardlink*, *rename*, *open*, *close*, *read*, *write*, *fsync*, *truncate*, *ftruncate*, *lseek*.

6.2 Dash

Dash is a differential abstract state hasher, implemented in Rust as a standalone binary. It is used to compare states between file systems, similar to how it's done in Metis [8]. There are some subtleties, though. For example, different file systems may use different block sizes, resulting in variations in file sizes. Another issue is that directory hard links are not supported by any major file system, and instead the hard link count can represent the number of files in directory – as is the case with Ext4, but not Btrfs, for instance. Because of this, some properties can be optionally ignored by changing configuration when required.

Dash represents the abstract state by calculating a hash for the entire file system, recursively processing each directory. All files are sorted by their relative paths. If files have different properties or if a file is missing in one file system, the hash will differ, and the fuzzer will detect this unexpected behavior. Additionally, the abstract state is saved in JSON format, allowing the root cause of the issue (e.g., a missing file) to be identified in the bug report.

6.3 DIFFuzzer

DIFFuzzer is the core component of the system. Its functionality is best explained by outlining the steps it performs from start to finish.

At the startup DIFFuzzer launches a QEMU [17] instance, using a preconfigured Linux OS image. The image must have a kernel built with custom configuration that enables KCov coverage collection and kernel memory bug detection. The image should also have necessary libraries and packages installed, such as *make* and a C++ compiler. For convenience, we provide detailed documentation on setting up QEMU and Linux images.

Communication with VM is handled via the QEMU Monitor Protocol (QMP) [22] and SSH [23]. QMP is used to receive events about the VM state and jobs in progress, as well as to save and load snapshots. SSH is used to execute remote commands and copy files between host and VM.

DIFFuzzer waits a fixed amount of time for the VM to boot. It then copies the testing runtime and compiles it. After this a VM snapshot is saved using QMP. The snapshot serves as a clean state that is used to restore the VM whenever it panics or reboots, avoiding the need to set up everything from scratch.

At this point fuzzer is ready to begin generating and executing workloads. The fuzzer starts with an empty corpus, containing an empty workload. The corpus is a collection of seeds – workloads that are considered interesting, for example, if their execution covered new code paths. Optionally, the corpus can be loaded from a directory with tests generated from previous runs to speed up state exploration.

At the start of each fuzzing iteration the fuzzer picks a random seed from the corpus. There are many ways to choose seeds. Most primitive way is to use a queue and pick seeds in a circle. However, this is not very effective, and a different strategy is used in our implementation. As explained in the AFLFast paper [24] an efficient coverage-based grey-box fuzzer prioritizes inputs that have not been fuzzed very often and inputs that exercise low-frequency paths. This approach allows for faster exploration of newly discovered code regions.

After workload is picked, random mutations are applied to it. Mutated workload is translated to a C program and copied to VM. The program is then compiled and run by a test harness.

The test harness is the component of the system responsible for running tests. First it sets up a file system by mounting it, then the test is executed and at the end the file system is unmounted. Additionally, other components can observe or change the state before and after each execution using callbacks. If the VM panics or reboots during execution, the state is restored from the previously saved snapshot. To speed up execution, file systems are created on a special storage device – block RAM device (BRD) [15], provided by a standard loadable Linux kernel module.

Each test is executed for both file systems. Each run can result in one of three outcomes: successful execution, a timeout after a fixed period, or a VM panic or reboot. If both runs complete successfully then fuzzer compares the traces and abstract states of two file systems, reporting a bug if they are different. Otherwise, no additional bug detection is performed, and the timeout or panic is reported. If no unexpected behavior is observed, the workload is evaluated by a feedback mechanism. If executing the workload covers new code paths, it is added to corpus for future use.

The cycle repeats: a new workload is selected, mutated, executed, and analyzed.

Reduction is not performed during fuzzing, as it is time-consuming and fuzzing already suffers from inherently long execution times. Instead, test cases can be reduced in a separate mode by reading them from saved JSON files.

6.4 Usage and extensibility

DIFFuzzer is not only a grey-box fuzzer, but it can also perform black-box fuzzing, launch single tests for one or two file systems, and reduce found bug cases – all within a single tool. Most internal components are shared across these modes, which simplifies the architecture and improves robustness.

DIFFuzzer can be easily configured with a single configuration file. For example, the user can configure the number of mutations, mutation and operation weights, and workload length, among other options. DIFFuzzer is also easy to extend: new file systems can be added by just implementing a single Rust trait (interface). Support for seven kernel file systems (Ext4, Btrfs, XFS, F2FS, BcacheFS, JFS, NILFS2) and one FUSE-based [4] file system (LittleFS [25]) was added with minimal effort.

For DIFFuzzer using we've prepared detailed documentation, manuals and script examples.

7. Evaluation

We evaluated DIFFuzzer on a host machine running Ubuntu 22.04 and Ubuntu 24.04 server cloud images inside a VM with an 8-core Intel Core i7-10700 2.90GHz CPU and 32 GB RAM. A single VM uses 2 virtual cores and 4 GB RAM.

Most of the processor time is spent executing tests. Within a single VM, it is possible to execute approximately 2.5 tests per second using non-instrumented kernel and about 1.5 tests per second with instrumented kernel (with Kcov and sanitizers enabled) on both file systems, tested on Ext4 and Btrfs. These numbers vary across file systems; test execution for a single file system is roughly twice as fast.

```
#include "executor.h"

int fd_0;

void test_workload()
{
    do_mkdir("1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
    fd_0 = do_create("1/2", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
    do_remove("1/2");
    do_remove("1"); // Directory not empty(39)
    do_close(fd);
}
```

Listing 1. Test where LittleFS is unable to remove directory with open file.

We discovered three specification violation bugs in the FUSE [4] file system LittleFS [25] in fewer than 1000 executions. The first bug (Listing 1) occurs when LittleFS fails to remove a directory if a file within it is open, even if *unlink* function was called successfully before.

```
#include "executor.h"

int fd_0, fd_1;

void test_workload()
{
    do_create("/1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
    fd_0 = do_open("/1");
    do_rename("/1", "/2");
    // "do_write" writes 10 bytes from a buffer filled
    // with random data starting from index 0
    do_write(fd_0, 0, 10);
    do_close(fd_0);
    fd_1 = do_open("/2");
    do_read(fd_1, 10); // 0 bytes read, expected 10
}
```

Listing 2. Test where file content is lost when renaming open file before writing.

Index, Command, ReturnCode, Errno, Extra	Index, Command, ReturnCode, Errno, Extra
0, CREATE, 4, Success (0),	0, CREATE, 4, Success (0),
1, OPEN, 4, Success (0),	1, OPEN, 4, Success (0),
2, RENAME, 0, Success (0),	2, RENAME, 0, Success (0),
3, WRITE, 10, Success (0),	3, WRITE, 10, Success (0),
4, CLOSE, 0, Success (0),	4, CLOSE, 0, Success (0),
5, OPEN, 4, Success (0),	5, OPEN, 4, Success (0),
6, READ, 10, Success (0), hash=2fd...	6, READ, 0, Success (0), hash=1

Listing 3. Execution traces generated for Ext4 (left) and LittleFS (right).

The second bug (Listing 2) appears when data is written to a file using an open file descriptor; if the file was renamed prior to the write, the data is silently lost. This can be seen from comparing traces (Listing 3).

```
#include "executor.h"

int fd_0;

void test_workload()
{
    do_create("/1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
    fd_0 = do_open("/1", O_RDWR);
    // fills file with 64 zero bytes
    do_ftruncate(fd_0, 64);
    // `SEEK_END` sets descriptor offset to LENGTH + N
    do_lseek(fd_0, 0, SEEK_END); // offset is 0, expected 64
}
```

Listing 4. Test where descriptor file length is not updated after truncation.

The third bug (Listing 4) is triggered by truncating a file (extending file with zeros) with an open file descriptor and causes the file descriptor to use old file length instead of new. All bugs were detected using differential fuzzing by comparing LittleFS behavior against Ext4, which allows DIFFuzzer to identify inconsistencies that most other tools may miss. The bugs are reported on the project GitHub repository [26-28].

Several bugs were also discovered in a proprietary file system, which was one of the main goals of this research. These issues, primarily related to specification violations, were identified through differential fuzzing. All bugs were confirmed by the developers and subsequently fixed. Unfortunately, we don't have permission to share more details about the results.

After two months of intermittent testing – totaling about 10 million executions – DIFFuzzer did not discover any bugs when comparing Ext4 against Btrfs (Linux 5.15) and Ext4 against BcacheFS (Linux 6.14). One possible explanation is insufficient time; since fuzzing is inherently time-consuming, it may require tens or hundreds of millions of executions to uncover a single hidden bug. Another possible reason is that we used the latest available kernel versions, in which many previously reported bugs may have already been fixed. This could also explain why recent work such as Metis [8] failed to identify bugs in Ext4 and Btrfs, even after billions of executions. These results suggest a need to improve workload mutation strategies and enhance error detection mechanisms.

Ext4 was selected as the default reference file system due to its extensive testing history and relatively high performance. However, other file systems can also be used as reference targets.

In summary, the primary goal of this work has been achieved. However, further development is necessary. While the proposed approach successfully identifies bugs, additional efforts are required to improve the tool's effectiveness – whether by refining the current approach, optimizing performance, or incorporating techniques from related work.

8. Conclusion

In this paper, we present DIFFuzzer, a differential grey-box fuzzer designed to find bugs in file system implementations. It detects memory errors, logic errors, and specification violations, all of which may compromise the correct functioning of operating systems and applications. Unlike existing tools for file system testing, DIFFuzzer combines differential and grey-box fuzzing. This allows it to detect behavioral inconsistencies deep within the execution paths of two file systems, without requiring a formal specification model. In our experiments, DIFFuzzer discovered bugs in the evaluated file systems. Some of these bugs are difficult or impossible to detect using most existing tools. Therefore, we believe that DIFFuzzer represents a valuable contribution to file system development and testing. It is worth noting that we did not find bugs in widely used file systems like Ext4 and Btrfs. However, these results also highlight the tool's limitations and indicate the need for further improvements.

References

- [1]. (2015) Syzkaller: coverage-guided kernel fuzzer. Google. Accessed: 2025-03-29. [Online]. Available: <https://github.com/google/syzkaller>.
- [2]. S. Kim, M. Xu, S. Kashyap, J. Yoon, W. Xu, and T. Kim, "Finding semantic bugs in file systems with an extensible fuzzing framework," in Proceedings of the 27th ACM Symposium on Operating Systems Principles, ser. SOSP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 147–161. [Online]. Available: <https://doi.org/10.1145/3341301.3359662>.
- [3]. KCov: code coverage for fuzzing. The Linux Foundation. Accessed: 2025-03-29. [Online]. Available: <https://www.kernel.org/doc/html/latest/dev-tools/kcov.html>.
- [4]. FUSE. The Linux Foundation. Accessed: 2025-03-29. [Online]. Available: <https://www.kernel.org/doc/html/latest/filesystems/fuse.html>.
- [5]. S. Schumilo, C. Aschermann, R. Gawlik, S. Schinzel, and T. Holz, "kAFL: Hardware-Assisted feedback fuzzing for OS kernels," in 26th USENIX Security Symposium (USENIX Security 17). Vancouver, BC:

- USENIX Association, Aug. 2017, pp. 167–182. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/schumilo>.
- [6]. Afl. Google. Accessed: 2025-03-29. [Online]. Available: <https://github.com/google/AFL>.
- [7]. (2013) Intel processor trace. Intel Corporation. Accessed: 2025-03-29. [Online]. Available: <https://software.intel.com/en-us/blogs/2013/09/18/processor-tracing>.
- [8]. Y. Liu, M. Adkar, G. Holzmann, G. Kuenning, P. Liu, S. A. Smolka, W. Su, and E. Zadok, “Metis: File system model checking via versatile input and state exploration,” in 22nd USENIX Conference on File and Storage Technologies (FAST 24). Santa Clara, CA: USENIX Association, Feb. 2024, pp. 123–140. [Online]. Available: <https://www.usenix.org/conference/fast24/presentation/liu-yifei>.
- [9]. Spin. Accessed: 2025-03-29. [Online]. Available: <https://spinroot.com/spin/whatispin.html>.
- [10]. J. Mohan, A. Martinez, S. Ponnappalli, P. Raju, and V. Chidambaram, “CrashMonkey and ACE: Systematically testing file-system crash consistency,” *ACM Trans. Storage*, vol. 15, no. 2, Apr. 2019. [Online]. Available: <https://doi.org/10.1145/3320275>.
- [11]. D. Chen, Y. Jiang, C. Xu, X. Ma, and J. Lu, “Testing file system implementations on layered models,” in Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ser. ICSE ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1483–1495. [Online]. Available: <https://doi.org/10.1145/3377811.3380350>.
- [12]. W. Liu and A.-I. A. Wang, “LFuzz: Exploiting locality-enabled techniques for file-system fuzzing,” in *Computer Security – ESORICS 2023*, G. Tsudik, M. Conti, K. Liang, and G. Smaragdakis, Eds. Cham: Springer Nature Switzerland, 2024, pp. 507–525.
- [13]. M. Xu, S. Kashyap, H. Zhao, and T. Kim, “KRace: Data race fuzzing for kernel file systems,” in 2020 IEEE Symposium on Security and Privacy (SP), 2020, pp. 1643–1660.
- [14]. “IEEE/Open Group standard for information technology–portable operating system interface (POSIX™) base specifications, issue 8,” IEEE/Open Group Std 1003.1-2024 (Revision of IEEE Std 1003.1-2017), pp. 1–4107, 2024.
- [15]. Using the RAM disk block device with linux. The Linux Foundation. Accessed: 2025-03-29. [Online]. Available: <https://www.kernel.org/doc/html/latest/admin-guide/blockdev/ramdisk.html>.
- [16]. T. Ridge, D. Sheets, T. Tuerk, A. Giugliano, A. Madhavapeddy, and P. Sewell, “SibylFS: formal specification and oracle-based testing for POSIX and real-world file systems,” in Proceedings of the 25th Symposium on Operating Systems Principles, ser. SOSP ’15. New York, NY, USA: Association for Computing Machinery, 2015, p. 38–53. [Online]. Available: <https://doi.org/10.1145/2815400.2815411>
- [17]. QEMU: a generic and open source machine emulator and virtualizer. Accessed: 2025-03-29. [Online]. Available: <https://www.qemu.org>.
- [18]. Kernel virtual machine. Accessed: 2025-03-29. [Online]. Available: <https://linux-kvm.org/page/MainPage>.
- [19]. (2016) TriforceAFL. NCC Group. Accessed: 2025-03-29. [Online]. Available: <https://github.com/nccgroup/TriforceAFL>.
- [20]. LCov. Accessed: 2025-03-29. [Online]. Available: <https://github.com/linux-test-project/lcov>.
- [21]. A. Fioraldi, D. C. Maier, D. Zhang, and D. Balzarotti, “Libafl: A framework to build modular and reusable fuzzers,” in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1051–1065. [Online]. Available: <https://doi.org/10.1145/3548606.3560602>.
- [22]. QEMU machine protocol. Accessed: 2025-03-29. [Online]. Available: <https://wiki.qemu.org/Documentation/QMP>.
- [23]. OpenSSH. OpenBSD Foundation. Accessed: 2025-03-29. [Online]. Available: <https://www.openssh.com>.
- [24]. M. Bohme, V.-T. Pham, and A. Roychoudhury, “Coverage-based greybox fuzzing as markov chain,” in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS ’16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1032–1043. [Online]. Available: <https://doi.org/10.1145/2976749.2978428>.
- [25]. (2017) LittleFS-FUSE. Accessed: 2025-03-29. [Online]. Available: <https://github.com/littlefs-project/littlefs-fuse>.
- [26]. LittleFS-FUSE issue tracker. Renaming open file causes written data to be lost. Accessed: 2025-04-11. [Online]. Available: <https://github.com/littlefs-project/littlefs-fuse/issues/78>.
- [27]. LittleFS-FUSE issue tracker. Removing directory with unlinked open file fails. Accessed: 2025-04-11. [Online]. Available: <https://github.com/littlefs-project/littlefs-fuse/issues/79>.
- [28]. LittleFS-FUSE issue tracker. Descriptor file length is not updated after truncation. Accessed: 2025-05-02. [Online]. Available: <https://github.com/littlefs-project/littlefs-fuse/issues/81>.

Информация об авторах / Information about authors

Вячеслав Максимович КОВАЛЕВСКИЙ – инженер Института прикладных компьютерных наук университета ИТМО. Сфера научных интересов: динамический и статический анализ программ, фаззинг.

Vyacheslav Maksimovich KOVALEVSKY – engineer of the Institute of Applied Computer Science of the ITMO University. Research interests: dynamic and static software analysis, fuzzing.

Валерий Владимирович КЕЧИН – инженер Института прикладных компьютерных наук университета ИТМО. Сфера научных интересов: динамический и статический анализ программ, фаззинг.

Valeriy Vladimirovich KECHIN – engineer of the Institute of Applied Computer Science of the ITMO University. Research interests: dynamic and static software analysis, fuzzing.

Владимир Михайлович ИЦЫКСОН – кандидат технических наук, доцент Института прикладных компьютерных наук университета ИТМО. Сфера научных интересов: статический и динамический анализ программ, верификация программного обеспечения, методы обнаружения дефектов в исходном коде, методы автоматизации тестирования программ.

Vladimir Mikhailovich ITSYKSON – Cand. Sci. (Tech.), associate professor of the Institute of Applied Computer Science of the ITMO University. Research interests: static and dynamic software analysis, software verification, methods for detecting defects in source code, methods for automating software testing.

DOI: 10.15514/ISPRAS-2025-37(4)-18



Evaluating Structural Complexity of Workflow Nets Modeling Asynchronous Agent Interactions

E. Zemlyanoy, ORCID: 0009-0007-3452-1698 eozemlyanoy@edu.hse.ru

R. Nesterov, ORCID: 0000-0002-4162-9070 rnesterov@hse.ru

HSE University,

11 Pokrovsky Bulvar, Moscow, 101000, Russia.

Abstract. The structure of a process model discovered from an event log of a multi-agent system often does not reflect the system architecture with respect to agent interactions. The existing conformance checking quality dimensions mainly evaluate the extent to which the behavior a discovered model corresponds to event sequences recorded in an event log. These behavioral dimensions might be insufficient to differentiate process models discovered from an event log of the same multi-agent system with respect to the independence of agents and the complexity of their interactions. In this work, we propose a theoretically grounded approach to measuring the structural complexity of a process model representing a multi-agent system with asynchronously interacting agents. We also report the key outcomes from a series of experiments to evaluate the sensitivity of the proposed approach to structural modifications in process models.

Keywords: multi-agent systems; asynchronous interaction; process mining; Petri nets; workflow nets; structural complexity.

For citation: Zemlyanoy E., Nesterov R. Evaluating Structural Complexity of Workflow Nets Modeling Asynchronous Agent Interactions. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025, pp. 47-68. DOI: 10.15514/ISPRAS-2025-37(4)-18

Acknowledgements. This work was supported by the Basic Research Program at the HSE University, Moscow (Russia).

Анализ структурной сложности сетей потоков работ для моделирования асинхронного взаимодействия агентов

Е.О. Земляной, ORCID: 0009-0007-3452-1698 <eozemlyanoy@edu.hse.ru>

Р.А. Нестеров, ORCID: 0000-0002-4162-9070 <rnesterov@hse.ru>

*Национальный исследовательский университет «Высшая школа экономики»,
Россия, 101000, Москва, Покровский бульвар, д. 11.*

Аннотация. Структура модели процесса, полученной на основе журнала событий многоагентной системы, часто не отражает архитектуру системы с точки зрения взаимодействий между агентами. Существующие метрики проверки соответствия в основном оценивают степень соответствия поведения обнаруженной модели последовательностям событий, зафиксированным в журнале. Однако такие поведенческие метрики могут оказаться недостаточными для того, чтобы отличить модели процессов, извлеченные из журнала одной и той же многоагентной системы, с учетом степени независимости агентов и сложности их взаимодействий. В данной работе предлагается теоретически обоснованный подход к измерению структурной сложности модели процесса, представляющей многоагентную систему с асинхронным взаимодействием агентов. Также представлены ключевые результаты серии экспериментов, направленных на оценку чувствительности предложенного подхода к структурным изменениям в моделях процессов.

Ключевые слова: многоагентные системы; асинхронное взаимодействие; синтез моделей процессов; сети Петри; сети потоков работ; структурная сложность.

Для цитирования: Земляной Е.О., Нестеров Р.А. Анализ структурной сложности сетей потоков работ для моделирования асинхронного взаимодействия агентов. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 47–68 (на английском языке). DOI: 10.15514/ISPRAS-2025-37(4)-18.

Благодарности: Работа выполнена при поддержке Программы фундаментальных исследований НИУ ВШЭ, Москва.

1. Introduction

Modern information systems perform a wide variety of operations recorded in event logs that help to track system performance, as well as to investigate and eliminate unwanted behavior. Event logs consist of ordered sequences of events called traces. Event logs are used in process mining to discover real process models with the help of various automated techniques [1-2]. Discovered process models reflect behavioral deviations from the reference models prepared at the early stages of the information system life-cycle. These deviations can be caused by different reasons, including, for example, infrastructural changes or unexpected user behavior.

Process models discovered from event logs can be represented in numerous notations: Petri nets, heuristic nets, causal nets, or Business Process Model and Notation (BPMN). In our work, we use Petri nets [3] – a well-known formalism for modeling the process control-flow which represents causal relations among actions recorded in an event log. In addition, Petri nets can serve as the mathematical basis for practice-oriented modeling notations, such as BPMN [4].

A wide variety of process discovery algorithms allow us to discover different process models from the same event log. In this case, the problem of selecting the most appropriate model arises. This is especially acute for models discovered from event logs of multi-agent systems with asynchronously interacting agents, as we show by the following example. Graph-theoretic characteristics, such as density, miss the architectural aspects of a discovered model, as they do not use the fact that the model structure should represent agent interactions.

Conformance checking [5] behavioral dimensions, fitness and precision, may also not be of great use for differentiating process models discovered from the event logs of a multi-agent system. Most of process discovery algorithms seek for the perfect fitness, when a discovered model can execute

all traces in an event log, and for the high precision, when the behavior of a model is narrowed to the traces in an event log.

Consider the following example. Figure 1a shows the Petri net model for a multi-agent system with two agents exchanging messages via two channels. This model perfectly fits the trace given in Table 1, as it can be fully replayed. Precision of this Petri net is 0.85 [6]. Figure 1b shows the other Petri net perfectly fitting the event log in Table 1, and its precision is also perfect, since it can execute only this trace. Such a Petri net can be discovered using, for example, Inductive miner [7].

What is more important, the structure of the Petri net in Fig. 1b hides the architecture of a multi-agent system with two interacting agents – they cannot be fully identified as subnets. In other words, the structure of the Petri net in Fig. 1a is more complicated than the of the Petri net in Fig. 1b with respect to the independence of agents.

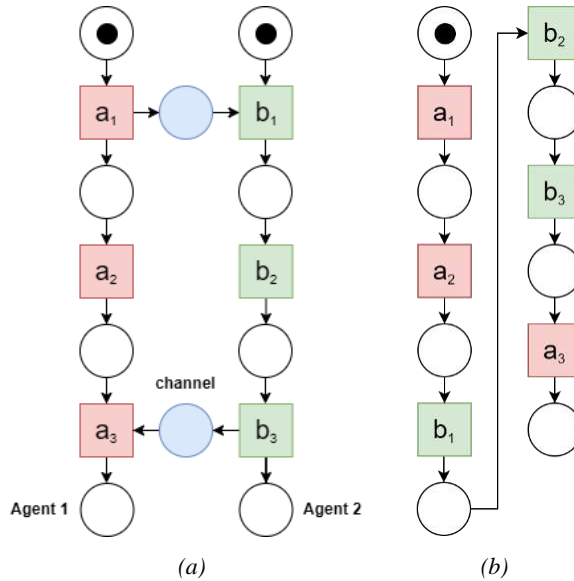


Fig. 1. Two models of the same multi-agent system.

This example highlights the problem that structural and behavioral conformance checking dimensions might be insufficient to evaluate the structural complexity of process models discovered from event logs of multi-agent systems. Moreover, the definitions of these dimensions vary across different approaches, as noted in [8], where the authors also stress the lack of the formal basis behind process model quality evaluation.

In this paper, we aim to propose a theoretically grounded approach to evaluating the structural complexity of process models discovered from event logs of multi-agent system, such that the independence of agents and the complexity of their asynchronous interactions are taken into account.

Table 1. The trace in an event log of a multi-agent system.

Timestamp	Action	Agent
2025-03-25 14:07:46.580465000	a ₁	Agent 1
2025-03-25 14:07:47.392390000	a ₂	Agent 1
2025-03-25 14:07:48.683925000	b ₁	Agent 2
2025-03-25 14:07:49.485992000	b ₂	Agent 2
2025-03-25 14:07:50.349081000	b ₃	Agent 2
2025-03-25 14:07:51.218921000	a ₃	Agent 1

The main contributions of our paper are:

- 1) A collection of structural complexity dimensions measuring the relative independence of agents in the structure of a multi-agent system model.
- 2) The key properties of the proposed dimensions, including compositionality and monotonicity with respect to the size of compared models.
- 3) The results of the experimental study aimed to assess the proposed approach sensitivity to structural modifications in process models of multi-agent systems.

The remainder of our paper is organized as follows. In the following section, we outline the theoretical background concerning Petri nets and the concept of neighboring transitions. Section 3 describes our approach to evaluating the structural complexity of process models based on various methods to aggregate and average the number of neighboring transitions. In Section 4, we highlight the key properties of our approach and report the main conclusions from a series of experiments. In Section 5, we discuss the related research, and Section 6 concludes the paper by emphasizing the main results and future work directions.

2. Preliminaries

In this section, we collect the basic definitions concerning Petri nets and their structure based on [3]. A restricted class of Petri nets is used in our paper to model multi-agent systems and asynchronous communications among agents. We also present the concept of neighboring transitions – the basis for evaluating the structural complexity of a multi-agent system model.

2.1 Workflow Nets

Workflow (WF) nets are extensively used to model the structure of processes discovered from event logs of information systems [9]. A WF-net is a restricted class of a Petri net.

Definition 1 [3]. A Petri net is a triple $N = (P, T, F)$, where P is a set of places, T is a set of transitions, $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation.

Graphically, a Petri net is shown as a bipartite graph, where places are circles, transitions are boxes, and the flow relation is shown by directed arcs. Places represent states, whereas transitions – actions executed by the agents in a multi-agent system. Figure 2 shows the example of a Petri net modeling the multi-agent system with two cyclic agents exchanging messages via channel places a and b .

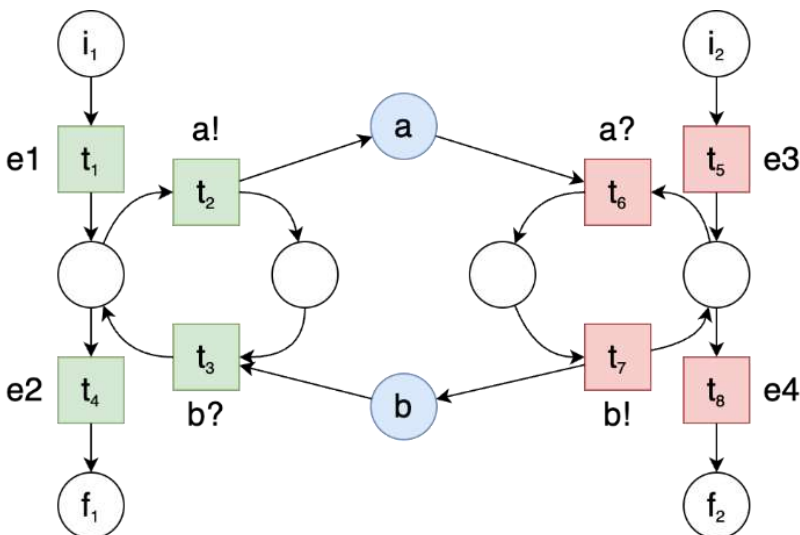


Fig. 2. Two asynchronously communicating agents.

Definition 2 [3]. Let $N = (P, T, F)$ be a Petri net. For any node $x \in P \cup T$:

- 1) $\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$ is the preset of x ;
- 2) $x \bullet = \{y \in P \cup T \mid (x, y) \in F\}$ is the postset of x .

The \bullet -notation can be extended to a set of nodes $X \subseteq P \cup T$ in a Petri net $N = (P, T, F)$, i.e., $\bullet X = \bullet x$ and $X \bullet = x \bullet$ for all $x \in X$. A subnet of N generated by $X \subseteq P \cup T$ is denoted $N(X) = (P \cap X, T \cap X, F \cap (X \times X))$. We consider Petri nets without self-loops, i.e., $\bullet x \cap x \bullet = \emptyset$, and isolated transitions, i.e., $|\bullet t| \geq 1$ and $|t \bullet| \geq 1$ for all $t \in T$.

A WF-net has the distinguished initial and final state represented by the corresponding subsets of places.

Definition 3 [9]. A WF-net $N = (P, T, F, in, fin)$ is a Petri net equipped with the initial state $in \subseteq P$ and the final state $fin \subseteq P$, where:

- 1) $\bullet in = fin \bullet = \emptyset$;
- 2) $\forall x \in P \cup T \exists i \in in \exists f \in fin: (i, x), (x, f) \in F^*$,

where F^* is the reflexive transitive closure of F .

Intuitively, according to the second requirement of this definition, every node in a WF-net lies on a path from its initial to its final state. The Petri net shown in Fig. 2 is also the WF-net, where the initial and final states of two communicating agents can be easily identified.

We assign labels to transitions in a WF-net. Transition labels correspond to actions recorded in the event log of a multi-agent system. Thus, we define a labeled WF-net.

Definition 4. Let A be a finite set of actions. A labeled WF-net $N = (P, T, F, in, fin, \lambda)$ is a WF-net equipped with a total transition labeling function $\lambda: T \rightarrow A \cup \{\tau\}$, where τ is the label of the silent action.

$T_{VIS} = \{t \in T \mid \lambda(t) \neq \tau\}$ denotes the set of visible transitions. Silent actions can also be referred to as invisible which means that they are internal system actions and are not assigned to an agent. Apart from that, silent actions are also used in process mining algorithms to organize the structure of a model discovered from an event log.

For example, the labels of the transitions in the WF-net shown in Fig. 2 are provided next to the transitions. We will further work only with labeled WF-nets. Therefore, we will omit the term “labeled”, and we will not clutter figures with transition labels. By convention, transitions corresponding to silent actions will be shown by black boxes.

In our study, we consider multi-agent systems with k asynchronously communicating agents. Thus, the set of actions A in Definition 4 can be partitioned into k subsets, i.e., $A = \{A_1, A_2, \dots, A_k\}$, where A_i with $i = 1, \dots, k$ is the set of actions executed by the specific agent.

This partition of A naturally induces the partition of transitions in a WF-net $N = (P, T, F, in, fin, \lambda)$ into k subsets with respect to their labels. Consequently, $T = \{T_1, T_2, \dots, T_k\}$, where $T_i = \{t \in T \mid \lambda(t) \in A_i\}$ with $i = 1, \dots, k$.

By convention, we will use colors to distinguish transitions belonging to different subsets in the partition of T . For example, the green and red colors distinguish two agents in the WF-net shown in Fig. 2.

2.2 Composing Labeled WF-nets via Channel Places

To model the asynchronous interactions among agents in a multi-agent system, we use the composition of WF-nets with labeled transitions studied in [10]. This operation involves inserting channel places between specifically labeled transitions:

- 1) label “c!” – sending a message to channel “c”;
- 2) label “c?” – receiving a message from channel “c”.

In Fig. 2, the channel place “a” connects two transitions with complement labels (“a!” and “a?”). The same holds for the channel place “b”. By convention, channel places are colored blue. The composition of two WF-nets, denoted by $N_1 \otimes N_2$, is also a WF-net. In addition, this operation is both commutative and associative. Thus, it is a convenient tool to model multi-agent systems with three or more interacting agents.

Our paper is focused on the structural analysis of multi-agent system models. Thus, we do not recall the behavior-preserving properties of the asynchronous WF-net composition proven in [10].

2.3 Neighboring Transitions

The main intuition behind our approach to evaluating the structural complexity of a multi-agent system model, given by a WF-net, lies in identifying the structural connections among transitions representing actions executed by different agents. In this light, we adopt the notion of neighboring transitions discussed earlier in [11].

Definition 5. Let $N = (P, T, F, in, fin, \lambda)$ be a WF-net. Neighboring transitions are defined by a relation $nbt \subseteq T \times T$, where $\forall t_1, t_2 \in nbt$ if and only if:

- 1) $\lambda(t_1) \neq \tau$ and $\lambda(t_2) \neq \tau$;
- 2) $\forall t \in T \{t_1, t_2\}: (t_1, t), (t, t_2) \in F^* \Rightarrow \lambda(t) = \tau$

If $(t_1, t_2) \in nbt$, we refer to t_2 as the neighbor of t_1 . The second requirement in Definition 5 tells that transition t_2 is the neighbor of t_1 if and only if all other transitions lying on a path from t_1 to t_2 are silent. In addition, the nbt relation is not necessarily symmetric, since there can be different paths connecting t_1 and t_2 in both directions (see Fig. 3, where silent transitions are shown by black boxes).

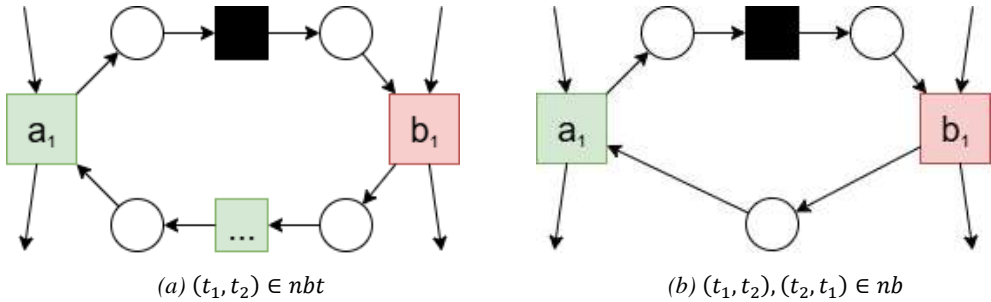


Fig. 3. Neighboring transitions are not necessarily symmetric.

Next, using the nbt relation, we identify the neighbors of transition $t \in T$ in a WF-net with respect to the partition of T induced by the labeling.

Definition 6. Let $N = (P, T, F, in, fin, \lambda)$ be a WF-net, where $T = \{T_1, T_2, \dots, T_k\}$ according to λ . For $t \in T_i$ with $i = 1, \dots, k$ define:

- 1) $all(t) = \{t' \in T \mid (t, t') \in nbt\}$;
- 2) $diff(t) = \{t' \in all(t) \mid t' \notin T_i\}$

Let $T_{WN} = \{t \in T \mid all(t) \neq \emptyset\}$ denote the subset of transitions that have neighbors.

The set $all(t)$ contains all neighbors of transition t according to nbt . The set $diff(t)$ includes only those neighbors that correspond to actions executed by agents different from that of t according to the labeling. In Fig. 2, for example, transition t_2 labeled by “a!” has $all(t_2) = \{t_3, t_6\}$ and $diff(t_2) = \{t_6\}$. Note that transitions t_4 and t_6 have $all(t_4) = all(t_6) = \emptyset$, since they both have the single outgoing place not connected to other transitions.

In the next section, we explore the structural complexity of the WF-net representing a multi-agent system with k asynchronously interacting agents with the help of neighboring transitions discussed above.

3. Structural Complexity of WF-nets

In this section, we present our approach to evaluating the structural complexity of WF-nets modeling multi-agent systems with asynchronously interacting agents. The approach is based on the identification of neighboring transitions. Intuitively, the number of neighboring transitions $diff(t)$ across a complete WF-net reflects the extent to which different agents are mutually independent.

We say that the structure of a WF-net reflects the interaction-oriented viewpoint of the multi-agent system architecture if the aggregated number of neighboring transitions $diff(t)$ is minimized to those directly involved in the asynchronous agent interactions via channel places. Given a WF-net $N = (P, T, F, in, fin, \lambda)$, we introduce the general notion of Neighbor Independence $NI(N)$ – a $[0, 1]$ -normalized value with respect to the following two principles for all $t \in T_{WN}$:

- 1) p0: $NI(N) = 0 \Leftrightarrow diff(t)$ is equal to $all(t)$;
- 2) p1: $NI(N) = 1 \Leftrightarrow diff(t)$ contains no transitions.

Figure 4 illustrates the main intuition behind these principles, which we discuss further in more detail.

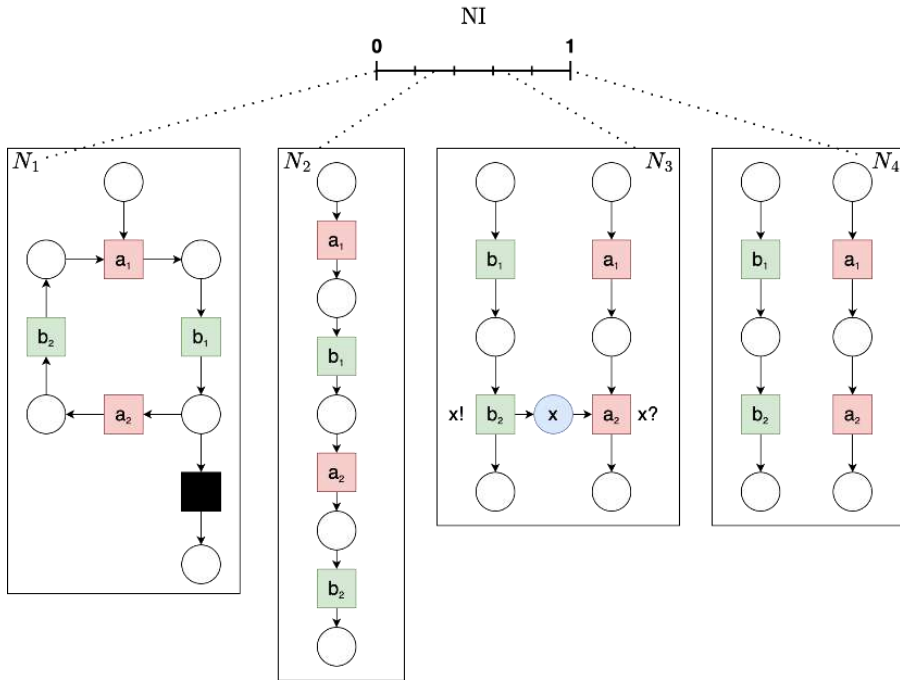


Fig. 4: The basic principles of Neighbor Independence.

Table 2 provides the configuration of the neighboring transitions for WF-nets N_1, N_2, N_3 and N_4 shown in Fig. 4. These WF-nets represent four different models of the multi-agent system with two asynchronously communicating agents. Each agent executes two actions. In this table, we construct a couple $\langle all(t), diff(t) \rangle$ for every transition in the mentioned WF-nets.

The first principle p0 corresponds to a WF-net whose structure does not allow us to distinguish individual agents. Such structure does not reflect the interaction-oriented viewpoint of the multi-agent system architecture. For example, in N_1 , shown in Fig. 4, transitions corresponding to the

behavior of different agents alternate. N_2 slightly differs from N_1 , since there is a single transition b_2 in N_2 without neighbors.

Table 2. Neighboring transitions for WF-nets from Fig. 4.

	a_1	a_2	b_1	b_2
N_1	$\langle\{b_1\}, \{b_1\}\rangle$	$\langle\{b_2\}, \{b_2\}\rangle$	$\langle\{a_2\}, \{a_2\}\rangle$	$\langle\{a_1\}, \{a_1\}\rangle$
N_2	$\langle\{b_1\}, \{b_1\}\rangle$	$\langle\{b_2\}, \{b_2\}\rangle$	$\langle\{a_2\}, \{a_2\}\rangle$	$\langle\emptyset, \emptyset\rangle$
N_3	$\langle\{a_2\}, \emptyset\rangle$	$\langle\emptyset, \emptyset\rangle$	$\langle\{b_2\}, \emptyset\rangle$	$\langle\{a_2\}, \{a_2\}\rangle$
N_4	$\langle\{a_2\}, \emptyset\rangle$	$\langle\emptyset, \emptyset\rangle$	$\langle\{b_2\}, \emptyset\rangle$	$\langle\emptyset, \emptyset\rangle$

The second principle p1 corresponds to a WF-net with completely isolated agents (see WF-net N_4 in Fig. 4). This is also demonstrated by the emptiness of $diff(t)$ for every transition in N_4 as given in Table 2. Multi-agent systems with isolated agents and, correspondingly, with the perfect value of NI should be considered as the degenerate case, since the usual architecture of a system involves agent interactions.

In other cases, the value of $NI(N)$ should be strictly between 0 and 1, according to the number of (channel) places connecting transitions corresponding to actions executed by different agents. For example, N_3 shown in Fig. 4 does not fall directly into p0 or p1.

In the following paragraphs, we propose different methods for aggregating and averaging the ratios of $diff(t)$ to $all(t)$ for all transitions in a WF-net in order to obtain the $[0, 1]$ -normalized value. If we detect “pathological” cases violating the basic principles p0 and p1, we identify the main causes and consider how to handle these cases.

3.1 Global Neighbor Independence

In this paragraph, we present our first attempt to evaluate the structural complexity of a multi-agent system model. Global Neighbor Independence directly evaluates the aggregated ratio of $|diff(t)|$ to $|all(t)|$ for all transitions in a WF-net N .

Definition 7. Let $N = (P, T, F, in, fin, \lambda)$ be a WF-net. The Global Neighbor Independence $GNI(N)$ is calculated as follows:

$$GNI(N) = 1 - \frac{\sum_{t \in T} |diff(t)|}{\sum_{t \in T} |all(t)|} \tag{1}$$

Table 3 provides the GNI values computed for the four WF-nets N_1, N_2, N_3 and N_4 from Fig. 4 as well as for the WF-net shown in Fig. 2. It can be clearly seen that the ordering of these values generally conforms with the basic principles p0 and p1 of Neighbor Independence discussed earlier.

Table 3. Comparison of GNI values.

	N_1	N_2	N_3	Fig. 2	N_4
$GNI(\cdot)$	0	0	0.667	0.833	1

We should note that comparing neighbor connectivity values of WF-nets representing completely different multi-agent systems is useless. For instance, $GNI(N) > GNI(N_2)$, where N is the WF-net from Fig. 2, but the overall structure of N is perceived to be more complex in comparison with N_2 from Fig. 4 at least because of the number of channel places.

The ordering $GNI(N_1) < GNI(N_2) < GNI(N_3) < GNI(N_4)$ is easy to explain, since the structure of these WF-nets demonstrates different levels of mutual independence of two asynchronously communicating agents. GNI does not distinguish N_2 from N_1 , since the absence of neighbors for b_2 in N_2 cannot influence the fraction in (1).

We can obtain $GNI(N)$ close to 1 for a WF-net N with a lot of channel places connecting transitions of different agents. The following proposition and example shown in Fig. 5 illustrates this problem.

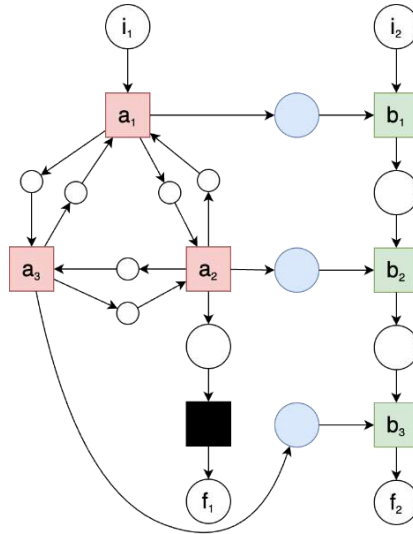


Fig. 5. A pathological WF-net with GNI close to 1.

The main idea behind this WF-net is that there are transitions, such that $diff(t)$ represents a tiny portion of $all(t)$ making the fraction in Definition 7 close to 0.

Proposition 1. There exists a labeled WF-net N with $GNI(N)$ tending to 1, such that the principle p1 is violated.

Proof. Consider a WF-net $N = (P, T, F, in, fin, \lambda)$ shown in Fig. 5. This WF-net models asynchronous interactions between two agents through three channels.

Transitions in the leftmost (red) agent form the cyclic structure, where every transition is the neighbor to the others. The rightmost (green) agent has the simple sequential structure with the same number of visible transitions as the red agent. If we remove channel places from this WF-net, $GNI(N)$ will be 1, since there are isolated agents.

Let us introduce channels as shown in Fig. 5, where every transition in the first agent connected via a single channel place to a single transition in the second agent. We will have that $GNI(N) = 8/11$ which is close to 1, but the agents are tightly connected. In addition, we generalize this case to n transitions in each agent:

- $X = \sum_{t \in T} |diff(t)| = n$ and
- $Y = \sum_{t \in T} |all(t)| = n(n - 1) + 2n - 1 = n^2 + n - 1$

Taking the value of n to infinity, we obtain that the ratio X/Y (required by Definition 7) tends to 0, and GNI tends to 1. Thus, we come to the violation of the principle p1, since it implies isolated agents, but the structure of N with many connections between agents reflects the exact opposite situation. We obtain the exaggerated value of GNI for a WF-net when there are too many internal neighbors within the same agent. The relative impact of inserting a new channel place between a pair of transitions, i.e., increasing $|diff(t)|$, can be negligible. We call this the neighbor explosion problem.

A possible immediate solution to this problem could be obtained by evaluating the global ratio of internal transitions of agents to all visible transitions in a WF-net, as formalized in the following definition.

Definition 8. Let $N = (P, T, F, in, fin, \lambda)$ be a WF-net. Let $T_{IN} = \{t \in T \mid diff(t) = \emptyset\}$. The Perimeter of Neighbor Independence $PNI(N)$ is calculated as follows:

$$PNI(N) = \frac{|T_{IN}|}{|T_{VIS}|} \tag{2}$$

Consider the values of PNI provided in Table 5 for the WF-nets shown in Fig. 2, Fig. 4 and Fig. 5. Firstly, we see that PNI resolves the pathological case of GNI shown in Fig. 5 by collapsing the value to 0.5 as soon as Formula 2 eliminates the impact of the local neighbor structure of the red agent. Secondly, it is clear that PNI conforms with the basic principles p0 and p1.

Table 4. Comparison of PNI values.

	N_1	N_2	Fig. 5	Fig. 2	N_3	N_4
$PNI(\cdot)$	0	0.250	0.500	0.750	0.750	1

Note that PNI has the locality property in the sense that changes in the structure of a WF-net do not require the complete recalculation as opposed to GNI. If we insert (remove) a channel place connecting internal transitions of different agents in a WF-net N , we only need to subtract (add) $1/|T|$ to (from) the current value $PNI(N)$. For example, if we insert a new channel between transitions b_1 and a_1 in N_3 shown in Fig. 4, the new value $PNI(N_2) = 0.75 - 0.25 = 0.5$.

However, inserting a new channel between two transitions of different agents that already have neighbors in the other agent does not affect the value of PNI. For example, if we insert a new channel place between transitions b_2 and a_1 in N_3 from Fig. 4, PNI remains the same (0.75), since T_{IN} does not change. Thus, the disadvantage to PNI is the lack of sensitivity to changes introduced into the way of agent interactions. In general, we consider PNI to be a partial improvement of GNI for the resolution of the neighbor explosion problem.

To sum up our reasoning on Global Neighbor Independence, it is necessary to highlight the special WF-net for which the denominator in (1) becomes 0. This WF-net represents a multi-agent system with k isolated agents – each agent can execute only a single action modeled by a single transition, respectively (see Fig. 6).

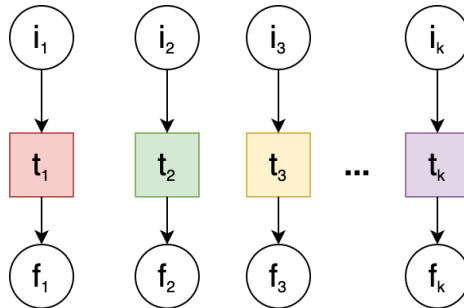


Fig. 6. A WF-net with single-transition isolated agents.

In order to overcome this zero-denominator problem, we need to exclude transitions without neighbors with $all(t) = \emptyset$ from the consideration. That is why Definition 6 introduces the set $T_{WN} = \{t \in T \mid all(t) \neq \emptyset\}$.

3.2 Local Neighbor Independence

This paragraph studies the averaged version of GNI. In fact, for each transition t in a WF-net N , we evaluate the ratio of $diff(t)$ to $all(t)$ and aggregate them separately as formalized below. Recall that T_{VIS} denotes the subset of visible transitions whose label is different from τ .

Definition 9. Let $N = (P, T, F, in, fin, \lambda)$ be a WF-net. Local Neighbor Independence $LNI(N)$ is calculated as follows:

$$LNI(N) = 1 - \frac{1}{|T_{VIS}|} \sum_{t \in T_{WN}} \frac{|diff(t)|}{|all(t)|} \quad (3)$$

Note that LNI does not take transitions without neighbors into account. Intuitively, these transitions do not make a meaningful contribution to the complexity of a WF-net representing a multi-agent system with asynchronously interacting agents.

Consider the values of LNI computed for WF-nets shown in Fig. 2, Fig. 4 and Fig. 5. Overall, we see that LNI does not violate the basic principles p0 and p1, as shown by the ordering of the four WF-nets in Fig. 4.

Table 5. Comparison of LNI values.

	N_1	N_2	N_3	Fig. 5	Fig. 2	N_4
$LNI(\cdot)$	0	0.250	0.750	0.833	0.896	1

LNI is sensitive to all changes that can be introduced into a WF-net N regarding the way agents communicate as opposed to GNI and PNI . The corresponding change in the value of LNI can be easily calculated. Let N' be a new WF-net obtained after inserting a new channel place connecting transition t_1 to transition t_2 in N , such that t_1 and t_2 do not belong to the same set in the partition of T according to λ . Then, $LNI(N') = LNI(N) + \Delta$, where Δ is:

$$\frac{1}{T_{VIS}} \left(\frac{|diff(t_1)|}{|all(t_1)|} - \frac{|diff(t_1)+1|}{|all(t_1)+1|} \right) \quad (4)$$

For example, let us insert a channel place from transition a_1 to transition b_1 in N_3 shown in Fig. 4, i.e., make b_1 a neighbor to a_1 . The direct application of Definition 9 gives us $LNI(N') = \frac{5}{8} = 0.625$. The change Δ , calculated by (4), is $1/4(0/1 - 1/2) = -1/8 = -0.125$. Therefore, $LNI(N') = LNI(N_3) + \Delta = 0.750 - 0.125 = 0.625$, which conforms with Definition 9 applied directly.

Similarly, we can derive the formula for the change in the value of LNI corresponding to the removal of a channel place connecting transition t_1 to transition t_2 in N , i.e., decreasing the number of neighbors for t_1 , such that t_1 and t_2 do not belong to the same set in the partition of T according to λ . Then, $LNI(N') = LNI(N) + \Delta$, where Δ is:

$$\frac{1}{T_{VIS}} \left(\frac{|diff(t_1)|}{|all(t_1)|} - \frac{|diff(t_1)-1|}{|all(t_1)-1|} \right) \quad (5)$$

For example, let us remove a channel between transitions a_1 and b_1 in the WF-net N shown in Fig. 5. Note that the original $LNI(N) = 5/6 = 0.833$. The direct application of Definition 9 gives us $LNI(N') = 16/18 = 0.889$. The corresponding change Δ , calculated by (5), is $1/6(1/3 - 0/2) = 1/18$. Thus, $LNI(N') = LNI(N) + \Delta = 5/6 + 1/18 = 16/18$, which again agrees with Definition 9 applied directly.

Overall, (4) and (5) help eliminate the need to recalculate LNI for a whole WF-net, once changes to the structure of agent communication are introduced.

Next, we discuss the key example to differentiate LNI from GNI . Consider WF-nets N_{MT} and N_{ST} shown in Fig. 7. They represent two ways of asynchronous interactions between two agents via a fixed number of channels using multiple transitions and a single transition, respectively. N_{ST} cannot be obtained using the composition described in Section 2.2.

By Definition 7, $GNI(N_{MT}) = GNI(N_{ST}) = 1 - 3/7 = 0.571$. Global Neighbor Independence cannot distinguish the inner structure of asynchronous agent interactions, since the effect of $diff(t)$ is spread among all other neighbors. However, by Definition 9, $LNI(N_{MT}) = 0.667 < LNI(N_{ST}) = 0.875$. Local Neighbor Independence not only aggregates and averages the number of transition neighbors, but also reveals the details on the asynchronous interaction.

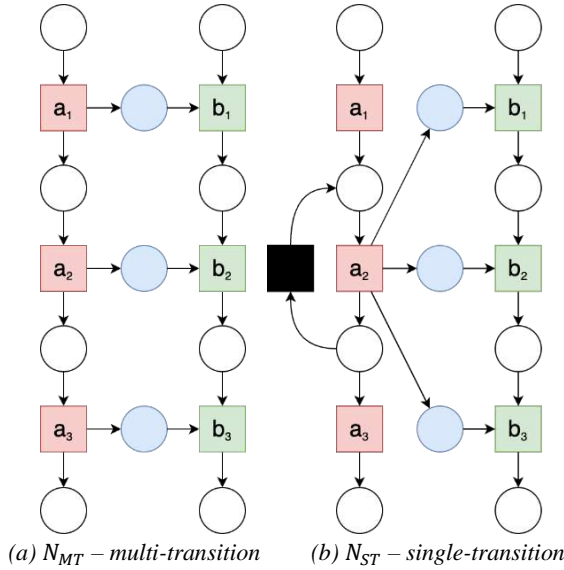


Fig. 7. Asynchronous interactions via several channels.

Thus, decreasing the number of transitions involved in asynchronous interactions, while maintaining the same number of channel places, makes the structure of model simpler and increases the value of *LNI*. Consider the generalization of WF-nets shown in Fig.7 up to n transitions in each agent:

$$LNI(N_{MT}) = 1 - \frac{1}{2n} \left(\frac{n-1}{2} + 1 \right) = \frac{3n-1}{4n} \tag{6}$$

$$LNI(N_{ST}) = 1 - \frac{1}{2n} \left(\frac{n}{n+1} \right) = \frac{2n+1}{2n+2} \tag{7}$$

It is easy to see that $LNI(N_{MT}) < LNI(N_{ST})$ for $n > 1$.

To sum up, *LNI* does not solve the neighbor explosion problem (see Fig. 5 and Proposition 1). The localized calculation of $|diff(t)|/|all(t)|$ does not help us to avoid tending *LNI* to 1, since each transition in the leftmost agent has too many neighbors, whereas only one of them is connected with the rightmost agent. We can try to “shift” the weight of the aggregated fraction $|diff(t)|/|all(t)|$ in order to move the value *LNI* to the left of 1 – a possible way for this will be discussed in the following paragraph.

Nonetheless, the advantages to Local Neighbor Independence, discussed above, outweigh the problem with the pathological WF-net.

3.3 Balanced Local Neighbor Independence

Recall that the set of visible transitions T_{VIS} in a WF-net N are partitioned into k subsets – $T_{VIS} = \{T_1, T_2, \dots, T_k\}$, where T_i is the subset of transitions corresponding to the actions executed by the specific agent with $i = 1, 2, \dots, k$.

Here, we consider the use of a weighted average of *LNI*’s computed separately for each agent in a WF-net. In order to keep the notation simple, the definition provided below implicitly considers transitions only with neighbors, where $all(t) \neq \emptyset$.

Definition 10. Let $N = (P, T, F, in, fin, \lambda)$ be a WF-net, where $T_{VIS} = \{T_1, T_2, \dots, T_k\}$ is a partition of T_{VIS} according to λ . Let w_1, w_2, \dots, w_k be weights, where $w_1 + w_2 + \dots + w_k = 1$. Balanced Local Neighbor Independence $BLNI(N)$ is calculated as follows:

$$BLNI(N) = 1 - \sum_{i=1}^k \frac{w_i}{|T_i|} \sum_{t \in T_i} \frac{|diff(t)|}{|all(t)|} \tag{8}$$

The main question with *BLNI* is how to arrange weights w_1, w_2, \dots, w_k . On the one hand, weights can be assigned equally, i. e., $w_1 = w_2 = \dots = w_k = 1/k$. In this case, every agent has the equal contribution to the structural complexity of a WF-net. On the other hand, the uneven arrangement of weights can be used to increase (decrease) the relative impact of an agent on the structural complexity of a WF-net.

Table 6 provides the values of *BLNI* for the pathological WF-net from Fig. 5 with different arrangement of agent weights, where w_R is the weight for the leftmost (red) agent and w_G is the weight for the rightmost (green) agent. Since we want to increase the impact of the red agent, we show how the value of *BLNI* changes with the increase in the weight w_R .

As we can see in Table 6, increasing the weight w_R of the red agent leads to a decrease of the value of *BLNI* up to its minimum when $w_R = 1$. However, this cannot completely resolve our pathological WF-net example in Fig. 5 in the general case. Therefore, we can admit that localization of the structural complexity evaluation does not overcome the neighbor explosion problem when there are transitions for which $|diff(t)|/|all(t)|$ tends to 0.

Table 6. *BLNI* with different weight arrangements for the WF-net from Fig. 5.

	$w_R = 0.5$ $w_G = 0.5$	$w_R = 0.7$ $w_G = 0.3$	$w_R = 1$ $w_G = 0$
<i>BLNI</i> (.)	0.833	0.767	0.667

Another possible reason to assign uneven weights w_i to agents in (8) is the significant difference in their size (the number of transitions). Consider the WF-net shown in Fig. 8, where the agent sending a message has a single transition, while the other agent has eight transitions. The values of *LNI* and *BLNI* with various weight arrangements for this WF-net are given in Table 7 below, where w_R is the weight of the red agent, and w_G is the weight of the green agent.

As we can see in Table 7, the equal weight assignment for agents in the WF-net in Fig. 8 significantly lowers the structural complexity evaluation compared to *LNI*. The main reason for this is that the only transition in the green agent is involved into the communication with the red agent. In addition, we note that the arbitrarily arranged weights ($w_G = 0.7, w_R = 0.3$) that ignore, for instance, size ratio, can result in the poorly interpreted result.

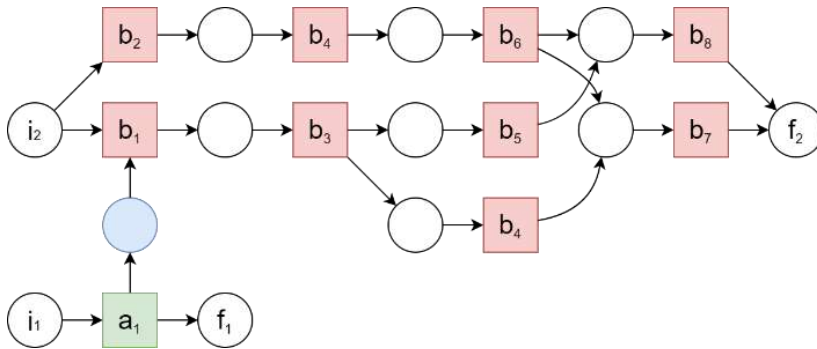


Fig. 8. A WF-net with significantly different agents.

Thus, according to the values provided in Table 7, *LNI* gives the most appropriate evaluation of the structural complexity of a multi-agent system with two agents (with a very different size) communicating via a single channel.

To sum up, *BLNI* is a flexible version of *LNI*, since one can assign different weights to agents. However, weights assigned to agents should not conflict with the real structure of a WF-net. Otherwise, one can dramatically change the original *LNI*. For example, if we set $w_G = 0$ and $w_R =$

1 for the pathological WF-net N in Fig. 5, we obtain $BLNI(N) = 0$, but the structure of this WF-net allows us to identify asynchronously communicating agents as subnets.

Table 7. LNI and BLNI with different weight arrangements for the WF-net from Fig. 8.

$LNI(.)$	$BLNI(.)$		
	$W_R = 0.5$ $W_G = 0.5$	$W_R = 0.7$ $W_G = 0.3$	$W_R = 1$ $W_G = 0$
0.900	0.500	0.300	0.700

Therefore, the basic recommendation for $BLNI$ can be either to use the standard equal weight arrangement $w_1 = w_2 = \dots = w_k = 1/k$ or to use an arrangement scheme that agrees with the internal structure of agents, for instance, based on the number of transitions. For the equal weight arrangement $w_1 = w_2 = \dots = w_k = 1/k$, the following obvious property, connecting LNI and $BLNI$, holds.

Proposition 2. Let $N = (P, T, F, in, fin, \lambda)$ be a WF-net, where $TVIS = \{T_1, T_2, \dots, T_k\}$ is the partition of T according to λ , and $|T_1| = |T_2| = \dots = |T_k|$. Let $w_1 = w_2 = \dots = w_k = 1/k$ be the weights. Then $BLNI(N) = LNI(N)$.

Proof. Substitute the given weights into (8):

$$BLNI(N) = 1 - \sum_{i=1}^k \frac{1}{k|T_i|} \sum_{t \in T_i} \frac{|diff(t)|}{|all(t)|} \tag{9}$$

As soon as $|T_1| = |T_2| = \dots = |T_k|$:

$$BLNI(N) = 1 - \frac{1}{|TVIS|} \sum_{i=1}^k \sum_{t \in T_i} \frac{|diff(t)|}{|all(t)|} \tag{10}$$

Propagate the sum over i in (10):

$$BLNI(N) = 1 - \frac{1}{|TVIS|} \sum_{t \in T_{WN}} \frac{|diff(t)|}{|all(t)|} = LNI(N)$$

since Definition 10 does not consider transitions without neighbors.

3.4 Neighbor Independence: Discussion

Here, we summarize the main aspects of our approach to measuring the structural complexity of a multi-agent system model in terms of the extent to which the structure of a WF-net reflects the interaction-oriented viewpoints of the system architecture. We introduced Neighbor Independence to evaluate how tight or loose the internal structure of asynchronous agent interactions in a WF-net is. Various methods of aggregating and averaging neighboring transitions gave rise to the following notions:

- 1) Global Neighbor Independence (GNI);
- 2) Perimeter of Neighbor Independence (PNI);
- 3) Local Neighbor Independence (LNI);
- 4) Balanced Local Neighbor Independence (BLNI).

Based on the analysis performed in the previous paragraphs, we compare these variations of Neighbor Independence against the following criteria:

- **c1:** conformance with the basic principles;
- **c2:** sensitivity to the communication structure;
- **c3:** sensitivity to changes in the communication structure;
- **c4:** locality of recalculation.

The criterion $c1$ also considers the neighbor explosion problem, which is the main reason for the violation of the principle $p1$. Table 8 provides the concise comparison protocol.

Table 8: Comparison of the various versions of Neighbor Independence.

	<i>GNI</i>	<i>LNI</i>	<i>BLNI</i>	<i>PNI</i>
c1	partial			yes
c2	no	yes	yes	yes
c3	yes	yes	yes	no
c4	no	yes	yes	yes

Based on these comparison results, we consider *LNI* to be the best choice for evaluating the structural complexity of a multi-agent system model with respect to the way agents interact. The neighbor explosion problem can be considered as a degenerate modeling case which is very unlikely to occur. In the next section, we discuss the key properties associated with the calculation of *LNI* and conduct a series of experiments to confirm the main conclusions obtained here.

4. Properties and Experimental Evaluation

In this section, we discuss several key properties of our approach to evaluating the structural complexity of a WF-net, which represents a multi-agent system with k asynchronously interacting agents. Firstly, we demonstrate that:

- 1) *LNI* can be computed in a compositional way;
- 2) *LNI* is monotone with respect to the size of WF-nets connected by an abstraction/refinement relation.

Secondly, to analyze the behavior of Neighbor Independence concerning structural changes in WF-nets, we conduct a series of experiments and report the main outcomes from them.

4.1 Compositionality of LNI

If we partition the set of transitions T in a WF-net N and substitute this partition into (3), we obtain that $LNI(N)$ is a weighted average as proven in the following proposition, which considers the basic case of partitioning T into two subsets.

Proposition 3. Let $N = (P, T, F, in, fin, \lambda)$ be a labeled WF-net. Let $T_{VIS} = T_1 \cup T_2$, where $T_1 \cap T_2 = \emptyset$. Then:

$$LNI(N) = w_1 LNI(N_1) + w_2 LNI(N_2)$$

where N_i is the subnet of N generated by T_i and $w_i = |T_i| / (|T_1| + |T_2|)$ with $i = 1, 2$.

Proof. The proof is by construction. By Definition 9 for N :

$$LNI(N) = 1 - \frac{1}{|T_1| + |T_2|} \sum_{t \in T_1} \sum_{t \in T_2} \frac{|diff(t)|}{|all(t)|} \quad (11)$$

Expand the sum over transitions with neighbors in (11):

$$LNI(N) = 1 - \frac{1}{|T_1| + |T_2|} \sum_{i=1}^2 \sum_{t \in T_i} \frac{|diff(t)|}{|all(t)|}$$

By Definition 9, for the subnet N_i with $i = 1, 2$:

$$LNI(N_i) = 1 - \frac{1}{|T_i|} \sum_{t \in T_i} \frac{|diff(t)|}{|all(t)|} \quad (12)$$

Multiply both sides in (12) by w_i with $i = 1, 2$:

$$w_i LNI(N_i) = w_i - \frac{1}{|T_1| + |T_2|} \sum_{t \in T_i^{WN}} \frac{|diff(t)|}{|all(t)|}$$

Since $w_1 + w_2 = 1$ by the proposition assumption, we have:

$$\sum_{i=1}^2 w_i LNI(N_i) = 1 - \frac{1}{|T_1| + |T_2|} \sum_{i=1}^2 \sum_{t \in T_i^{WN}} \frac{|diff(t)|}{|all(t)|}$$

which exactly corresponds to $LNI(N)$.

Thus, $LNI(N)$ corresponds to the average of $LNI(N_i)$ computed independently for the subnet N_i of N and weighed by the size of these subnets (the number of transitions T_i). In fact, Proposition 3 allows us to choose arbitrary subnets (possibly having common places) in a WF-net N to calculate $LNI(N)$ (see Fig. 9a, where subnets contain transitions belonging to different agents). However, we believe that the appropriate way to identify subnets in a WF-net N is to isolate transitions belonging to different agents, as shown in Fig. 9b, which corresponds to the composition of WF-nets as discussed in Section 2.2. This WF-net models a multi-agent system with three interacting agents, i.e., the set of transitions is partitioned into three subsets $T = \{T_1, T_2, T_3\}$, where $T_1 = \{a_1, c_1\}$, $T_2 = \{b_1\}$ and $T_3 = \{d_1\}$.

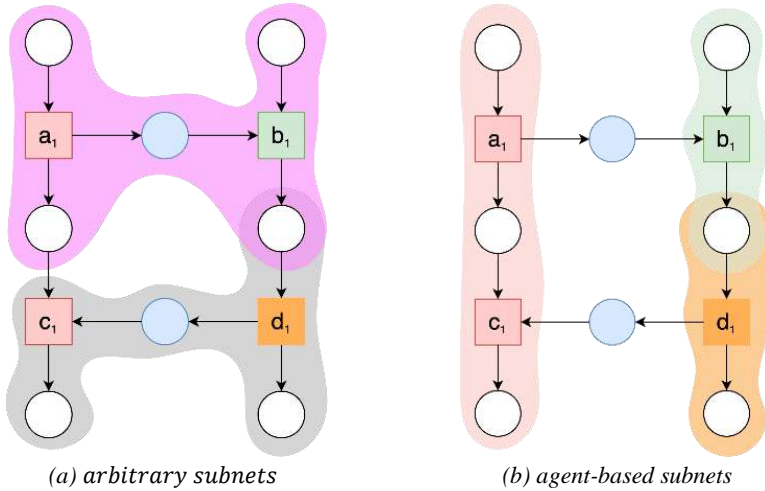


Fig. 9. Two ways of choosing subnets in a WF-net.

Now consider the calculation of $LNI(N)$ for the WF-net N in Fig. 9. The direct application of Definition 9 gives $LNI(N) = 1 - 1/4 (1/2 + 1/1 + 1/1) = 3/8 = 0.375$. Let us use the arbitrary subnets given by Fig. 9a to calculate $LNI(N)$, which, in this case, is equal to the following:

$$\frac{1}{2} \left(1 - \frac{1}{2} \left(\frac{1}{2} + \frac{1}{1} \right) \right) + \frac{1}{2} \left(1 - \frac{1}{2} \left(\frac{1}{1} \right) \right) = \frac{3}{8} \tag{13}$$

Let us also use the agent-based subnets given by Fig. 9b to calculate $LNI(N)$, which is equal to the following in this case:

$$\frac{2}{4} \left(1 - \frac{1}{2} \left(\frac{1}{2} \right) \right) + \frac{2}{4} \left(1 - \frac{1}{1} \left(\frac{1}{1} \right) \right) = \frac{3}{8} \tag{14}$$

Both (13) and (14) yield the same result corresponding to Definition 9 applied directly.

The compositionality behind Local Neighbor Independence allows us to decompose the calculation process for the large multi-agent system model into several independent parts that can be executed in parallel.

4.2 Monotonicity of LNI

As we mentioned in Section 3, comparing models of completely different multi-agent system with respect to Neighbor Independence is useless. However, in this paragraph, we aim to consider a restricted class of related models whose structural complexity comparison is meaningful. These are abstract WF-nets and their refinements related via α -morphisms [12]. The example of an α -morphism $\varphi: N_1 \rightarrow N_2$, where N_1 is a refinement of the abstract WF-net N_2 , is shown in Fig. 10. N_2 can be seen as an interface which is implemented by N_1 . In addition, both N_1 and N_2 can be seen as a composition of two WF-nets via two channels used to exchange messages – N_1 exhibits the refined behavior of the red agent. In fact, α -morphisms are extensively used in [11] to identify which structural and behavioral properties are preserved in the composition of WF-nets.

Intuitively, within an α -morphism $\varphi: N_1 \rightarrow N_2$, a place in N_2 is refined by a subnet in N_1 . For instance, a place p_2 in N_2 is refined (shown by the dashed arrow in Fig. 10) by a subnet with two transitions in N_1 .

An α -morphism $\varphi: N_1 \rightarrow N_2$ is a surjective map, fully defined by the way transitions in N_1 are mapped on transitions in N_2 . If transition t in N_1 is mapped to place p in N_2 , then $\bullet t \bullet$ is mapped to p as well. If transition t_1 in N_1 is mapped to transition t_2 in N_2 , then $\varphi(\bullet t_1) = \bullet t_2$ and $\varphi(t_1 \bullet) = t_2 \bullet$.

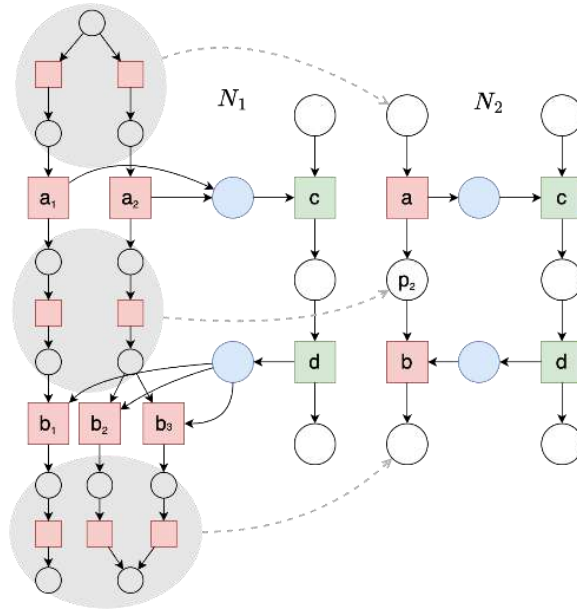


Fig. 10. The α -morphism between two WF-nets N_1 and N_2 .

As proven in the following proposition, Local Neighbor Independence is monotone with respect to the size of WF-nets related by an α -morphism under the certain restrictions.

Proposition 4. Let $N_i = (P_i, T_i, F_i, in_i, fin_i, \lambda_i)$ be a WF-net with $i = 1, 2$. Let $\varphi: N_1 \rightarrow N_2$ be an α -morphism, s.t. $|T_1 \cap \varphi^{-1}(P_2)| \geq |\varphi^{-1}(T_2)|$. Then, $LNI(N_1) \geq LNI(N_2)$.

Proof. The requirement $|T_1 \cap \varphi^{-1}(P_2)| \geq |\varphi^{-1}(T_2)|$ implies that the number of transitions in N_1 that are mapped on places in N_2 is at least as big as the number of transitions in N_1 that are mapped on transitions in N_2 . Since φ is surjective, $|T_1| \geq 2|T_2|$. By Definition 9, $LNI(N_1)$ is:

$$1 - \frac{1}{|T_1|} \sum_{t_1 \in T_1} \frac{|diff(t_1)|}{|all(t_1)|} \leq 1 - \frac{1}{2|T_2|} \sum_{t_1 \in T_1} \frac{|diff(t_1)|}{|all(t_1)|} \quad (15)$$

By Definition 9, $LNI(N_2)$ is:

$$1 - \frac{1}{|T_2|} \sum_{t_2 \in T_2} \frac{|diff(t_2)|}{|all(t_2)|} \quad (16)$$

Note that the value of the sum in (15) over the set T_1 is determined by those transitions in T_1 that are also mapped on transitions in N_2 , because, for each transition t_1 mapped to a place in N_2 , $|diff(t)| = 0$. There are at least $|T_2|$ transitions in N_1 mapped on transitions in N_2 .

In N_1 , there can be several cases concerning transitions mapped by φ on transitions. Firstly, $|diff(t_1)|$ can remain the same as $|diff(\varphi(t_1))|$ in N_2 , and $|all(t_1)|$ can increase by new internal neighbors or remain the same. Then, (15) \geq (16), which means $LNI(N_1) \geq LNI(N_2)$. Secondly, both $|diff(t_1)|$ and $|all(t_1)|$ can grow. Then, $|T_1| \geq a |T_2|$, where $a > 2$. Correspondingly, the value subtracted from 1 in (15) will be adjusted downward. Thus, $LNI(N_1) \geq LNI(N_2)$.

In other words, we show that a refined WF-net N_1 , obtained by substituting places for subnets, cannot be structurally worse than an abstract WF-net N_2 in terms of asynchronous agent interactions. A refined WF-net introduces more internal agent transitions and lowers the relative impact of connections to other agents. For instance, $LNI(N_1) = 0.857 > LNI(N_2) = 0.625$ for the α -morphism provided in Fig. 10. We admit that we impose the rather strong restriction on the size of N_1 , that can be weakened during the future research.

4.3 Experiment Layout

The behavior of Neighbor Independence concerning changes in WF-nets is analyzed using a family of randomly generated models. The experimental procedure includes:

- 1) Initialization. We specify parameters for the random WF-net generation:
 - o the number of agents;
 - o the number of places, transitions within each agent;
 - o the number of places used for agent interactions;
 - o the so-called density of arcs within each agent d_{in} and between them d_{out} .

In the context of our experiments, we work with agents of the same size (the number of transitions).

- 2) Generation. We generate the required number of places and transitions and randomly add arcs between them according to densities chosen in the previous step.
- 3) Evaluation. We calculate the values of GNI and LNI for the generated WF-nets. $BLNI$ is out of the scope of experiments, since it coincides with LNI for equally-sized agents (see Proposition 2).

Densities of arcs are defined by a number $0 \leq d_{in}, d_{out} \leq 1$ which represent the random discrete variable ξ_{in} and ξ_{out} – both of them is defined as shown in Table 9.

Table 9: Probability values for ξ_{in} and ξ_{out}

	$1 - d_{\{in/out\}}$	$d_{\{in/out\}}/2$	$d_{\{in/out\}}/2$
ξ	no arc between a place and a transition	arc from a place to a transition	arc from a transition to a place

Experiments are conducted to verify that:

- h1:** increasing the number of channels between unchanged
- h2:** increasing the number of places in agents with unchanged channels increases the values of GNI and LNI .

4.4 Experimental Results

Here, we report the key outcomes from the experiments conducted to analyze the behavior of *GNI* and *LNI* with respect to **h1** and **h2**. According to the layout described previously, we generated random WF-nets with $d_{in}, d_{out} = 0.8$ varying the number of transitions, places and channels. For each set of generation parameters, we produced and analyzed 100 WF-nets by averaging and assessing the standard deviation of *GNI* and *LNI*, respectively.

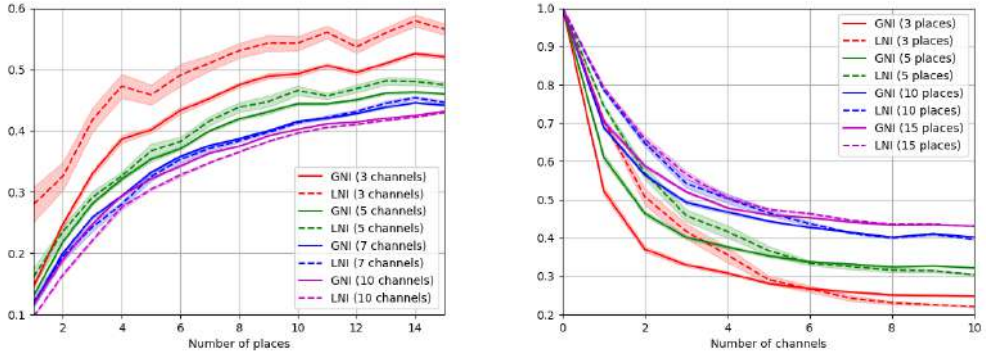


Fig. 11. A selection of experimental results for WF-nets with 5 transitions per agent.

The complete specification of the experimental results can be found in the repository [13], while Fig. 11 provides a selection of these results for WF-nets where each agent consists of 5 transitions. The shaded areas around the lines show the standard deviation.

Firstly, the left chart in Fig. 11 supports **h1**, showing a decrease in dimension values as the number of channels increases. Moreover, both *GNI* and *LNI* tend to similar values which rise as the number of internal agent places increases. Secondly, the right chart in Fig. 11 supports **h2**. The growth in the internal complexity of agent increases only the denominator in (1) and (3) leading to the respective increase of *GNI* and *LNI*, as shown in this chart.

To sum up, **h1** and **h2** follow from the way *GNI* and *LNI* are calculated according to Definition 7 and Definition 9. Moreover, complete experimental results shows that the trends in Fig. 11 are the same regardless of the size of agents.

5. Related work

As we mentioned in Introduction, there are many algorithms for the automated discovery of process models from event logs of information systems. Among the most wide-spread are Inductive miner [7], Fuzzy miner [14] and Region theory-based miner [15]. They overcome typical problems of event logs – incompleteness (a finite event log cannot cover all possible process scenarios) and noise (missing or duplicated activities). Inductive miner can also guarantee the perfect fitness and the behavioral correctness of a discovered WF-nets.

The existence of a large number of different process discovery algorithms makes the quality evaluation of discovered models a very important problem. Conformance checking [5] provides several basic quality dimensions – fitness, precision, generalization and simplicity. Fitness, precision and generalization are focused on measuring the correspondence between a discovered model and an event log from a behavioral point of view. Simplicity is aimed at assessing the structure of a discovered model, identifying, for example, useless nodes. Process models discovered from event logs of multi-agent systems requires special attention, since behavior-based quality dimensions may not differentiate models regarding the extent to which their structure reflects key interaction-oriented view- points of the multi-agent system architecture.

General graph-theoretic complexity measures, including density and cyclomatic complexity [16], consider the entire structure of a model, which is not sufficient for a model representing a multi-agent system with asynchronously inter-acting agents. Similarly, in [17], the authors discussed several empirical approaches to measuring the general complexity of Petri net. That is why, we developed the special approach to evaluating the structural complexity of a WF-net.

The problem of discovering process models from event logs with the understandable structure was considered in various contexts. A set of papers [18-20] proposed different approaches to improving the structure of discovered models by the localization of the event environment in a log and by composing fragments of the regular behavior with the rare and exceptional scenarios. Discovery of hierarchical process models, where a high-level event represents a subprocess, was studied in [21], [22], where the authors also considered sophisticated cyclic and concurrence relations between subprocesses. The identification of low-level and high-level events in an event log is a natural way to improve the structural representation of a process model. The paper [23] initialized a new direction of modeling and discovering object-centric Petri nets from event logs. Interactions of objects are represented through complex synchronizations which allow one to model consumption and production of objects of different types. Compositional discovery of behaviorally correct and architecture-aware process models, whose structure reflects individual agents and their interactions, from event logs of multi-agent systems was studied in [11].

Our study continues [11] in a way that we expand on the notion of neighboring transitions. Based on the number of places connecting transitions of different agents in a WF-net representing a multi-agent system, we proposed several methods for aggregating and averaging the number of neighboring transitions. These methods exhibit various levels of sensitivity to changes in the structure of asynchronous agent interactions.

6. Conclusion and Future Work

In this paper, we considered the problem of assessing the structural complexity of process models representing multi-agent systems with asynchronously interacting agents. Specifically, we worked with WF-nets that can be discovered from event logs recording the historical data on multi-agent system operation. The structural complexity of a WF-net was expressed through the notion of Neighbor Independence intuitively showing how loose the inter-agent connections are. Using this idea with the various levels of locality, we proposed three $[0, 1]$ -valued dimensions of structural complexity showing the extent to which the structure of a WF-net reflects the interaction-oriented viewpoints of the system architecture.

We formulated the neighbor explosion problem – a special pathological case when the proposed dimensions tend to 1, since there are too many internal neighboring transitions within an agent. This can be resolved by refusing to perform the localized analysis of agent interactions. However, we consider this pathology to be a degenerate case. Its occurrence is unlikely in process modeling practice.

The core part of our study was devoted to identifying the main properties of the proposed structural quality dimensions. We proved that Local Neighbor Independence possesses compositionality and monotonicity. The latter holds for WF-nets related by a special abstraction/refinement relation under certain restrictions on their size. Sensitivity of the proposed dimensions to structural WF-net modifications was analyzed through the series of experiments. The experimental results are in agreement with the formal definitions. Inserting new channel places has a greater impact on the value of a dimension than increasing the internal complexity of an individual agent. Future research will be focused on the following directions. Firstly, we suggest extending the applicability of the proposed approach to general Petri nets. Secondly, we plan to develop an algorithm for localizing process model fragments that contribute most to the complexity of agent interactions.

References

- [1]. W. van der Aalst, *Process Mining: Data Science in Action*. Springer Heidelberg, 2016. URL: <https://link.springer.com/book/10.1007/978-3-662-49851-4>.
- [2]. A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, F. M. Maggi, A. Marrella, M. Mecella, and A. Soo, Automated discovery of process models from event logs: Review and benchmark, *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 4, pp. 686–705, 2019. URL: <https://ieeexplore.ieee.org/document/8368306>.
- [3]. W. Reisig, *Understanding Petri nets. Modeling techniques, analysis methods, case studies*. Springer Heidelberg, 2013. URL: <https://link.springer.com/book/10.1007/978-3-642-33278-4>.
- [4]. A. Kalenkova, A. Burattin, and M. a. de Leoni, Discovering high-level BPMN process models from event data, *Business Process Management Journal*, vol. 25, pp. 995–1019, 2019. URL: <https://www.emerald.com/bpmj/article/25/5/995/256745/Discovering-high-level-BPMN-process-models-from>.
- [5]. J. Carmona, B. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking: Relating Processes and Models*. Springer Heidelberg, 2018. URL: <https://link.springer.com/book/10.1007/978-3-319-99414-7>.
- [6]. A. Adriansyah, J. Munoz-Gama, J. Carmona, B. van Dongen, and W. van der Aalst, Alignment based precision checking, in *Business Process Management Workshops*, ser. LNBIP, vol. 132. Springer Heidelberg, 2013, pp. 137–149. URL: https://link.springer.com/chapter/10.1007/978-3-642-36285-9_15.
- [7]. S. Leemans, D. Fahland, and W. van der Aalst, Discovering block-structured process models from event logs – a constructive approach, in *PETRI NETS*, ser. LNCS, J. Colom and J. Desel, Eds., vol. 7927. Springer Heidelberg, 2013, pp. 311–329. URL: https://link.springer.com/chapter/10.1007/978-3-642-38697-8_17.
- [8]. W. M. van der Aalst, Relating process models and event logs – 21 conformance propositions, in *ATAED@Petri Nets/ACSD*, 2018. [Online]. URL: <https://api.semanticscholar.org/CorpusID:49303433>
- [9]. W. van der Aalst, Workflow verification: Finding control-flow errors using petri-net-based techniques, in *Business Process Management: Models, Techniques, and Empirical Studies*, ser. LNCS, vol. 1806. Springer, Heidelberg, 2000, pp. 161–183. URL: https://link.springer.com/chapter/10.1007/3-540-45594-9_11.
- [10]. L. Bernardinello, I. Lomazova, R. Nesterov, and P. L., Soundness-preserving composition of synchronously and asynchronously interacting workflow net components, *Journal of Parallel and Distributed Computing*, vol. 179, Article 104704, 2023. URL: <https://www.sciencedirect.com/science/article/pii/S0743731523000679>.
- [11]. R. Nesterov, L. Bernardinello, I. Lomazova, and P. L., Discovering architecture-aware and sound process models of multi-agent systems: a compositional approach, *Software and Systems Modeling*, vol. 22, pp. 351–375, 2023. URL: <https://link.springer.com/article/10.1007/s10270-022-01008-x>.
- [12]. L. Bernardinello, E. Mangioni, and L. Pomello, Local state refinement and composition of elementary net systems: An approach based on morphisms, in *Transactions on Petri Nets and Other Models of Concurrency VIII*, ser. LNCS, vol. 8100. Springer, Heidelberg, 2013, pp. 48–70. URL: https://link.springer.com/chapter/10.1007/978-3-642-40465-8_3.
- [13]. E. Zemlyanoy, Structural complexity of workflow nets [Online]. URL: <https://github.com/egorz734/structural-complexity-of-workflow-nets>.
- [14]. C. Gunther and W. van der Aalst, Fuzzy mining – adaptive process simplification based on multi-perspective metrics, in *Business Process Management*, ser. LNCS, G. Alonso, P. Dadam, and M. Rosemann, Eds., vol. 4714. Springer Heidelberg, 2007, pp. 328–343. URL: https://link.springer.com/chapter/10.1007/978-3-540-75183-0_24.
- [15]. J. van der Werf, B. van Dongen, C. Hurkens, and A. Serebrenik, Process discovery using integer linear programming, in *PETRI NETS*, ser. LNCS, K. van Hee and R. Valk, Eds., vol. 5062. Springer Heidelberg, 2008, pp. 368–387. URL: https://link.springer.com/chapter/10.1007/978-3-540-68746-7_24.
- [16]. T. McCabe, A complexity measure, *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976. URL: <https://ieeexplore.ieee.org/document/1702388>.
- [17]. L. Gang Soo and Y. Jung-Mo, An empirical study on complexity metrics of petri nets, *Microelectronics Reliability*, vol. 32, no. 9, pp. 1215–1221, 1992. URL: <https://www.sciencedirect.com/science/article/pii/0026271492900610>.
- [18]. A. Kalenkova, A. Lomazova, and W. van der Aalst, Process model discovery: A method based on transition system decomposition, in *PETRI NETS*, ser. LNCS, G. Ciardo and E. Kindler, Eds., vol. 8489.

Springer Heidelberg, 2014, pp. 71–90.

URL: https://link.springer.com/chapter/10.1007/978-3-319-07734-5_5.

- [19]. A. Kalenkova and I. Lomazova, Discovery of cancellation regions within process mining techniques, *Fundamenta Informaticae*, vol. 133, pp. 197–209, 2014. URL: <https://journals.sagepub.com/doi/abs/10.3233/FI-2014-1071>.
- [20]. W. van der Aalst, A. Kalenkova, V. Rubin, and E. Verbeek, Process discovery using localized events, in *PETRI NETS*, ser. LNCS, R. Devillers and A. Valmari, Eds., vol. 9115. Springer Heidelberg, 2015, pp. 287–308. URL: https://link.springer.com/chapter/10.1007/978-3-319-19488-2_15.
- [21]. A. Begicheva, I. Lomazova, and R. Nesterov, Discovering hierarchical process models: an approach based on events partitioning, *Modeling and Analysis of Information Systems*, vol. 31, no. 3, pp. 294–315, 2024. URL: <https://www.mais-journal.ru/jour/article/view/1879>.
- [22]. A. Begicheva and I. Lomazova, Discovering high-level process models from event logs, *Modeling and Analysis of Information Systems*, vol. 24, no. 2, pp. 125–140, 2017. URL: <https://www.mais-journal.ru/jour/article/view/504>.
- [23]. W. van der Aalst and A. Berti, Discovering object-centric petri nets, *Fundamenta Informaticae*, vol. 175, pp. 1–40, 2020. URL: <https://journals.sagepub.com/doi/10.3233/FI-2020-1946>.

Information about authors

Егор Олегович ЗЕМЛЯНОЙ – стажер-исследователь научно-учебной лаборатории процессно-ориентированных информационных систем факультета компьютерных наук НИУ Высшая школа экономики, магистр по направлению «Программная инженерия» с 2025 г. Его научные интересы включают моделирование и анализ поведения распределенных информационных систем с применением сетей Петри и их различных расширений.

Egor Olegovich ZEMLYANOV – a research assistant at the Laboratory of Process-Aware Information Systems (PAIS Lab), HSE University (Moscow, Russia), MSc in Software engineering since 2025. Research interests: modeling and analysis of distributed information systems behaviors using Petri nets and their extensions.

Роман Александрович НЕСТЕРОВ – доцент департамента программной инженерии факультета компьютерных наук, заведующий научно-учебной лабораторией процессно-ориентированных информационных систем НИУ Высшая школа экономики, кандидат компьютерных наук НИУ Высшая школа экономики с 2022 г. Его научные интересы включают подходы к моделированию и анализу поведения сложно организованных информационных систем, с помощью сетей Петри, теорию категорий и общую теорию параллелизма.

Roman Alexandrovich NESTEROV – Cand. Sci. (Computer Science), Associate Professor at the Department of Software Engineering at the Faculty of Computer Science, Head of the Laboratory of Process-Aware Information Systems (PAIS Lab), HSE University (Moscow, Russia). His research interests include approaches to modeling and analyzing the behavior of complex information systems using Petri nets, category theory, and the general theory of parallelism.

DOI: 10.15514/ISPRAS-2025-37(4)-19



Relaxed Lazy Soundness Verification for Data Petri Nets

N.M. Suvorov, ORCID: 0000-0003-2871-9757 <nmsuvorov@hse.ru>

I.A. Lomazova, ORCID: 0000-0002-9420-3751 <ilomazova@hse.ru>

HSE University,

20, Myasnitskaya Ulitsa, Moscow, 101000, Russia.

Abstract. To represent a model that includes both data and resource perspectives, Data Petri nets could be used. In this formalism, each transition has a constraint that includes input and output conditions on variables. To stay within decidability, the conditions should not contain arithmetic operations, so the resources are usually represented as separate places. Existing correctness criteria such as easy, relaxed and lazy soundness could be adapted to resource-oriented Data Petri nets but deciding them requires solving a reachability problem that is known to have very high complexity even for classical Petri nets. In this paper, we propose a new correctness notion called relaxed lazy soundness that incorporates the main features of the aforementioned properties and that could be decided as a coverability problem, which is known to be less computationally complex than the reachability one. We provide an algorithm to verify this property, prove its correctness, and implement it in the existing soundness verification toolkit. The performance evaluation results confirm the applicability of the algorithm to process models of a moderate size. The algorithm could be used both for verification of resource-oriented models and for preliminary validation of arbitrary process models represented as Data Petri nets.

Keywords: data Petri net; verification of distributed processes with data; soundness verification; transition systems.

For citation: Suvorov N.M., Lomazova I.A. Relaxed Lazy Soundness Verification for Data Petri nets. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025, pp. 69-84. DOI: 10.15514/ISPRAS-2025-37(4)-19.

Acknowledgements. This study has been supported by the Basic Research Program at HSE University, Russia.

Проверка ослабленной ленивой бездефектности для сетей Петри с данными

Н.М. Суворов, ORCID: 0000-0003-2871-9757 <nmsuvorov@hse.ru>

И.А. Ломазова, ORCID: 0000-0002-9420-3751 <ilomazova@hse.ru>

*Национальный исследовательский университет «Высшая школа экономики»,
Россия, 101000, г. Москва, ул. Мясницкая, д. 20.*

Аннотация. Для представления модели, включающей как данные, так и ресурсы, можно использовать сети Петри с данными. Каждому переходу в таких сетях сопоставлено ограничение, включающее условия на входные и выходные значения переменных. Условия на входные значения определяют правила, при которых переход может сработать. Условия на выходные значения определяют правила, согласно которым осуществляется изменение значений переменных при срабатывании перехода. Чтобы оставаться в границах разрешимости, условия не должны содержать арифметические операции, ввиду чего ресурсы обычно представляются как отдельные позиции. Существующие критерии бездефектности, такие как простая, ослабленная и ленивая бездефектность, могут быть адаптированы к ресурсно-ориентированным сетям Петри с данными, но для их вычисления требуется решение проблемы достижимости, имеющей высокую вычислительную сложность даже для классических сетей Петри. В этой статье мы предлагаем новое свойство бездефектности, называемое ослабленной ленивой бездефектностью, которое включает в себя основные характеристики вышеупомянутых свойств и которое может быть определено путем решения проблемы покрытия, имеющей меньшую вычислительную сложность чем проблема достижимости. Мы представляем алгоритм для проверки данного свойства, доказываем его корректность и реализуем его в существующем наборе инструментов проверки бездефектности. Результаты оценки производительности подтверждают применимость алгоритма для верификации моделей среднего размера. Алгоритм можно использовать как для проверки ресурсно-ориентированных моделей, так и для предварительной проверки произвольных моделей, представленных сетями Петри с данными.

Ключевые слова: сети Петри с данными; верификация распределенных процессов с данными; верификация бездефектности; системы переходов.

Для цитирования: Суворов Н.М., Ломазова И.А. Проверка ослабленной ленивой бездефектности для сетей Петри с данными. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 69–84 (на английском языке). DOI: 10.15514/ISPRAS-2025-37(4)-19.

Благодарности. Данное исследование поддержано Программой фундаментальных исследований Национального исследовательского университета «Высшая Школа Экономики», Россия.

1. Introduction

A business process consists of a set of activities that are performed in a coordinated manner in an organizational and technical environment and allow a business goal to be achieved [1]. Analyzing business process models at the design stage allows us to identify inconsistencies and potential errors before implementing the process. Our paper focuses on a specific correctness criterion for process models, called soundness. Different types of soundness properties have been defined for business processes. The most well-known are classical [2], weak [3], generalized [4], relaxed [5], and lazy [6] soundness. The paper [7] presents a classification of the main soundness properties proposed for process models.

A classical approach to verifying soundness of a process model is to represent it as a Petri net and check the soundness property for this representation. However, in data-aware process models, where data influence the execution of the process, classical Petri nets are not appropriate. In such cases, different extensions of Petri nets are used, one of which is Data Petri nets (DPNs) [8], which we consider in this paper. Data Petri Nets extend place/transition nets with variables, adding guards that depend on the values of these variables and allowing transitions to update them. Consequently, each

state in a DPN is represented as a pair that includes a marking and a variable valuation (values assigned to all model variables).

DPNs may incorporate both data and resource perspectives. The resources, as in classical Petri nets, could be represented as individual places that may be unbounded and contain extra tokens in a final state (since any DPN with arithmetic conditions is Turing complete, as was proved in [9], it is a reasonable choice to represent resources as places but not as variables). For DPNs, an algorithm to verify the classical soundness (whether a process always terminates properly and whether each process activity occurs in at least one of its executions) was proposed in [10]. However, this property is too strong, making the model incorrect unless it is bounded and does not allow for additional tokens in the final state. This makes this definition not suitable for checking correctness of resource-oriented DPNs.

An example of a resource-oriented DPN is shown in Fig. 1. Here, the model represents a gambling process and contains a resource place p_3 that stores game tokens obtained by a gambler. The idea of gambling is simple: A gambler rolls a number from 0 to 100 and based on this number, he/she may win or lose tokens or leave them unchanged. A gambler may quit the game whenever he/she wants. The initial number of tokens is 3. This DPN is unbounded, since p_3 may store any number of game tokens. Thus, it is not sound according to the definition of classical soundness proposed in [9]. However, the model can be considered correct for representing a real gambling process. Note that the following properties hold for this model: (i) it has at most one token in the sink place o , (ii) there are some number of feasible executions that end at o (albeit with possibly several tokens left in p_3), (iii) each transition occurs in at least one feasible execution. This means that the model is relaxed [5] and lazy [6] sound.

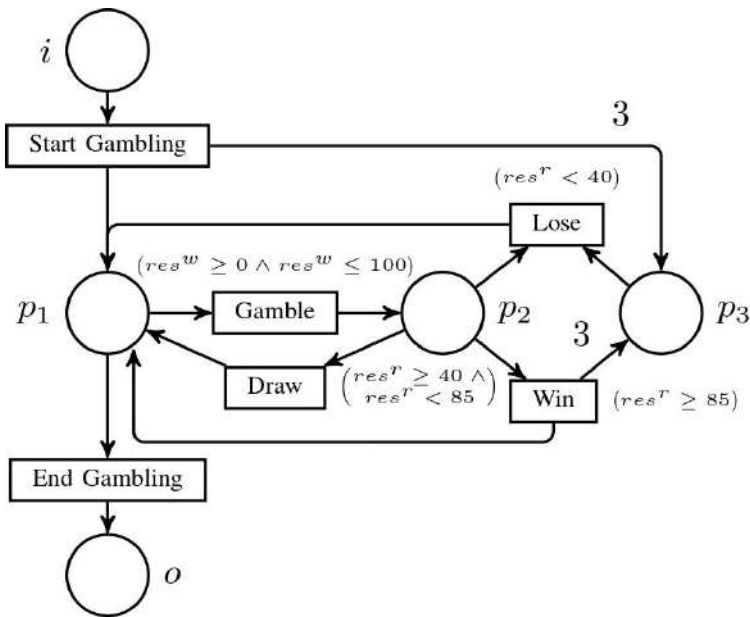


Fig. 1. Resource-oriented DPN \mathcal{N} representing a process of gambling. p_3 is a resource place representing game tokens.

Deciding relaxed and lazy soundness properties requires solving a reachability problem [7,11] that is known to have high time complexity even for classical Petri nets (a lower bound is a tower of exponentials [12]). Thus, these soundness properties can barely be checked in practice even for small models. For DPNs with variables of infinite domains, the task becomes even more complex. In this paper, we propose a new soundness property called relaxed lazy soundness, which combines the main features of relaxed and lazy soundness and deciding which can be reduced to solving a

coverability problem, which is significantly less computationally complex than the reachability one. Our property captures a possibility of termination, support for lazy activities [6], and a requirement of participation of all activities in process executions. Compared with lazy soundness, our property allows models to have deadlocks (as in relaxed soundness). Compared with relaxed soundness, our property supports lazy activities (as in lazy soundness). We propose an algorithm to verify relaxed lazy soundness, which, according to our experimental evaluation, can be used in practice for moderately sized process models.

2. Related Works

Much effort has been put into validating data-aware process models, both in terms of model checking and soundness verification. The approaches differ not only in the algorithms themselves but also in the formalisms used to represent such process models.

The main current standardized approach to represent data-aware process models is to use the Business Process Model and Notation (BPMN) together with the Decision Model and Notation (DMN) to achieve the separation of concerns while representing both control and data flows [13]. Some authors propose other BPMN extensions, such as BPMN models with conditions on arcs [14]. However, due to the complexity and ambiguity of the BPMN notation, simpler mathematical formalisms are usually used to represent data-aware process models and verify their correctness. A classical formal representation of a business process is a workflow net [15]. A workflow net is a Petri net that has a single source place (a place without input arcs), denoted i , and a single sink place (a place without output arcs), denoted o , where each node is on a path from i to o . Representing data-aware process models using workflow nets is a cumbersome task, as shown in [16]. The reason is that, for each variable value, a separate place should be generally added. This makes this approach applicable only for small process models with rather limited domains of variable values, as for larger models or domains such construction results in huge models that cannot be analyzed either manually or automatically.

Another approach is to use Petri net extensions. The authors in [17] propose a conceptual workflow model extended with data operations (WFD-net), where each model state is represented by a marking and a subset of satisfied guards. Here, the activities actually read/write entire guards, which may be not granular enough to represent real process models [18]. Paper [19] introduces a workflow model extended with SQL operations on tables (WFT-net) designed to overcome this limitation. In [20], this formalism is extended by adding domain constraints that must hold in each state of the model. Each state of the proposed formalism is represented as a tuple that includes a marking, a state of data elements, a state of tables, and a subset of satisfied guards. The number of elements that represent a state makes the state sufficiently difficult to perceive, while the data perspective appears rather abstract.

Our paper focuses on Data Petri nets proposed in [8]. In DPNs, the data perspective is represented by variable valuations, while each transition is accompanied by a constraint that includes input and output conditions on variables. Compared with WFD and WFT nets, most of the behavioral properties of DPNs are generally undecidable. Thus, to consider correctness properties, such as soundness, the authors usually consider a subclass of DPNs, where each variable is real-typed and conditions are composed of variable-operator-constant and variable-operator-variable atoms.

Regarding soundness properties, data-aware process models are verified mainly against classical soundness (for example, all soundness verification algorithms for DPNs, namely [18,21,22], check only classical soundness). Paper [7] proves that a Petri net is classical sound *if and only if* its closure (a workflow net, where a transition is added from i to o) is live (from any reachable marking it is possible to enable any transition) and bounded. However, in many scenarios, weaker correctness criteria are determined for models. For instance, in resource-oriented models, model unboundedness is not a sign of incorrectness and, thus, checking them against classical soundness is not meaningful. For such cases, weaker notions of soundness have been proposed. Let us consider some of them.

Probably, the most well-known property after classical soundness is *weak soundness* [3]. A model is weakly sound if and only if the process always properly terminates (i.e., leads to marking $[o]$). This notion allows dead transitions, which are never executed, but still requires the model to be bounded. For cases when it is important not to have dead transitions in the model, *relaxed soundness* [5] can be applied. A model is relaxed sound if and only if each model activity is present in some execution that leads to the final state. A relaxed sound model may be unbounded and may include deadlocks and livelocks. In resource-oriented models, it is often the case that proper termination (having a single token in o) is not required. This is the case where *lazy soundness* [6] can be applied: a process model is lazy sound if and only if the process always ‘adequately’ terminates. Here, the ‘adequate’ termination means reaching a marking with a single token in o and any number of tokens in other places, whereas markings having more than one token in o should not be reachable. Lazy soundness requires an ‘adequate’ termination from each reachable state, and thus deadlocks and livelocks are not allowed. Some authors also add other notions of soundness: for instance, [9] proposes the property (which they call ‘relaxed soundness’) that only requires the model to contain at least one proper execution. The last three soundness properties fit the case of resource-oriented models (indeed, all of them hold for the model from Fig. 1), but deciding each of them requires solving a reachability problem, which may take an unreasonable amount of time for unbounded models even of small sizes.

For DPNs, the first attempt to verify soundness was made in [10]. Here, the authors propose to convert a DPN into a colored Petri net (CPN) and then verify the soundness of the CPN using existing techniques. The authors restrict the DPN to contain only variable-operator-constant conditions. In all subsequent works, namely [21-23], the authors construct a finite abstraction of the state space, called either a Constraint Graph or a Labeled Transition System (LTS), to verify the classical soundness. Since, as proved in [22], such an abstraction of the state space is not sufficient by itself to verify the soundness property, various special techniques have been developed to verify soundness. In [23], the authors make an LTS as granular as possible by considering a graph node as a pair that includes a marking and the smallest set of variable assignments that can be described by a combination of atomic formulas present in the net. In [21], the authors construct a separate abstraction for each reachable DPN marking and then combine them. [22] modifies the original DPN by splitting transitions that belong to cycles and adding silent transitions in such a way that the resulting DPN is still equivalent to the original one, but its LTS can now be directly used to verify soundness.

All of the above algorithms are designed to verify the classical soundness. As mentioned in [23], they can also be used to verify weak soundness by skipping the step of detecting dead transitions. Checking classical and weak soundness can be done quick enough for small and medium-sized models as both of these properties require model boundedness. If a model is unbounded, solving a reachability problem becomes a significantly more complex task [24]. However, our investigation has shown that the property that combines the main features of relaxed and lazy soundness could be decided by a single coverability graph construction, which makes verification of this combined property a viable option for checking correctness of resource-oriented DPNs.

3. Data Petri Nets

A Data Petri net is a place/transition net that includes transition constraints, represented as logical expressions, that define input and output conditions over the data variables.

Each constraint φ over a set X of variables is an expression of the form

$$\varphi := \top \mid x \odot y \mid x \odot c \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2,$$

where: (i) \top is the logical “true”; (ii) $x, y \in X$; (iii) $c \in R$; (iv) $\odot \in \{<, =, >\}$.

By $\Phi(X)$, we denote the language of constraints. For example, for $X = \{y, z\}$, expressions $y > z$, $z < 2$ and $(z > 1) \vee ((z < 2) \wedge (y = 1))$ are in $\Phi(X)$.

Let X be a set of variables. A constraint $\varphi \in \Phi(X)$ is satisfied by an assignment $\theta: X \rightarrow \mathbb{R}$, written $\theta \models \varphi$, according to the following definitions:

- $\theta \models x \odot c$ if and only if $\theta(x)$ is defined, and $\theta(x) \odot c$ is true;
- $\theta \models x \odot y$ if and only if both $\theta(x)$ and $\theta(y)$ are defined, and $\theta(x) \odot \theta(y)$ is true;
- $\theta \models \neg\varphi$ if and only if $\theta \not\models \varphi$;
- $\theta \models \varphi_1 \wedge \varphi_2$ if and only if $\theta \models \varphi_1$ and $\theta \models \varphi_2$.

By $[[\varphi]]$, we denote the set of all assignments that satisfy $\varphi \in \Phi(X)$. We say that two formulas $\varphi_1, \varphi_2 \in \Phi(X)$ are logically equivalent (denoted $\varphi_1 \sim \varphi_2$) if and only if $[[\varphi_1]] = [[\varphi_2]]$.

Let V be some set of variables. Since in DPNs, variable values may be changed by a transition firing, for each variable $v \in V$, we use variables v^r (r – for ‘read’) and v^w (w – for ‘write’) to address its input and output (w.r.t. transition firings) values. Define $V^r \doteq \{v^r | v \in V\}$ and $V^w \doteq \{v^w | v \in V\}$. Then each transition constraint in a DPN is an expression in $\Phi(V^r \cup V^w)$.

Now we can define a DPN:

Definition 1 (Data Petri net). A *data Petri net* (DPN) is a tuple $\mathcal{N} = \langle P, T, F, V, guard \rangle$, where:

- P and T are disjoint sets of places and transitions, respectively;
- $F: (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ is a flow relation;
- V is a finite set of variables;
- $guard: T \rightarrow \Phi(V^r \cup V^w)$ is a guard assignment function that labels transitions with arithmetic constraints.

Given $t \in T$, we also define $read(t)$ and $write(t)$ to denote, respectively, the set of variables V^r and V^w that occur in $guard(t)$. In this work, we consider DPNs with distinguished input (denoted i) and output (denoted o) places.

A state of a DPN \mathcal{N} is a pair (M, α) , where

- $M: P \rightarrow \mathbb{N}$ is a marking function that assigns a number of tokens to each place $p \in P_{\mathcal{N}}$, and
- $\alpha: V \rightarrow \mathbb{R}$ is a variable valuation function that assigns a value to each variable in V .

We use $A_{\mathcal{N}}$ to denote the set of all possible variable valuations in \mathcal{N} and $\mathcal{M}_{\mathcal{N}}$ to denote the set of all markings in \mathcal{N} . Given two markings M' and M'' of a DPN \mathcal{N} , we write $M'' \geq M'$ if and only if for all $p \in P_{\mathcal{N}}$, we have $M''(p) \geq M'(p)$, and we write $M'' > M'$ if and only if $M'' \geq M'$ and there exists $p \in P_{\mathcal{N}}$ s.t. $M''(p) > M'(p)$.

A DPN changes from one state to another by firing transitions. Given a DPN \mathcal{N} and a state (M, α) , we say that transition $t \in T$ may fire at (M, α) yielding a new state (M', α') if and only if:

- $M(p) \geq F(p, t)$ and $M'(p) = M(p) - F(p, t) + F(t, p)$, for all $p \in P$;
- $\beta \models guard(t)$, where $\beta: V^r \cup V^w \rightarrow \mathbb{R}$ and, for every $v \in V$, it holds that $\beta(v^r) = \alpha(v)$ and $\beta(v^w) = \alpha'(v)$;
- $\alpha(v) = \alpha'(v)$, for every $v \in V$ such that $v^w \neq write(t)$.

We denote transition firing as $(M, \alpha)[t](M', \alpha')$.

This is naturally extended to finite sequences of transition firings $\sigma = t_1 \dots t_n$, called traces, while each trace induces a run denoted as $(M_0, \alpha_0)[t_1] \dots [t_n](M_n, \alpha_n)$ (or, equivalently, as $(M_0, \alpha_0)[\sigma](M_n, \alpha_n)$). Given two states (M, α) and (M', α') , we write $(M, \alpha)[*](M', \alpha')$ to denote zero or more transition firings leading from (M, α) to (M', α') . By (M_I, α_I) we denote the initial DPN state. As we consider resource-oriented nets, we assume that $M_I \geq [i]$ and $M_I[i] = 1$. For DPN \mathcal{N} with initial state (M_I, α_I) we define runs of \mathcal{N} and traces of \mathcal{N} as the set of runs and traces as above, of any length, such that $(M_I, \alpha_I)[\sigma](M, \alpha)$ for some marking M and variable valuation α .

Consider DPN \mathcal{N} in Fig. 1. For this net, $M_I = [i]$ and $\alpha_I(res) = 0$. Firing *Start Gambling* consumes a token from i and produces one token in p_1 and three tokens in p_3 . If now *End Gambling* fires, the process terminates. If *Gamble* fires, it transfers a token to p_2 and sets a value from 0 to 100 to variable res , based on which the choice between *Lose*, *Draw*, and *Win* is done on the next step. Note that *Lose* consumes one token from p_3 , *Win* produces three tokens in p_3 , and *Draw* does not change the number of tokens in p_3 . Firing any of these transitions adds a token to p_1 and another gambling attempt can be made. The process terminates when *End Gambling* fires or when a gambler loses all his game tokens.

Definition 2 (Reachability set, reachability graph). Let \mathcal{N} be a DPN with an initial state (M_I, α_I) . The reachability set of \mathcal{N} , denoted by $Reach_{\mathcal{N}}$, is the smallest set of states, which is inductively defined as follows:

- $(M_I, \alpha_I) \in Reach_{\mathcal{N}}$;
- if $(M, \alpha)[t](M', \alpha')$ for $t \in T$ and $(M, \alpha) \in Reach_{\mathcal{N}}$, then $(M', \alpha') \in Reach_{\mathcal{N}}$.

The reachability graph of \mathcal{N} , denoted as $RG_{\mathcal{N}}$, is a graph $\langle V, E \rangle$, where:

- $V = Reach_{\mathcal{N}}$ is the set of reachable states of \mathcal{N} ;
- $E \subseteq V \times T \times V$ is the set of edges such that $(v, t, v') \in E$ if and only if $v[t]v'$, for some $t \in T$.

Fig. 2 shows a fragment of $RG_{\mathcal{N}}$ for \mathcal{N} from Fig. 1. The fragment includes an execution that leads to the situation, when all tokens are lost (a path from s_0 to s_{10}), and a feasible execution that includes all DPN activities and ends in o (a path from s_0 to s_{16}). A special case where a gambler starts and immediately ends gambling is also added (a path from s_0 to s_2).

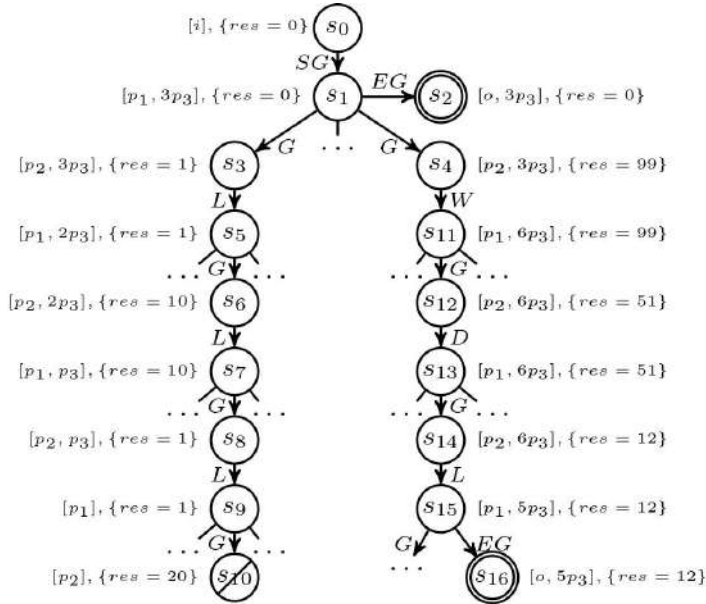


Fig. 2. A fragment of the reachability graph for \mathcal{N} from Fig. 1. Arcs are labeled with the initial letters of the transition names. Square brackets denote markings. Curly brackets denote variable valuations. Double circles denote final nodes. Forbidden signs denote deadlocks.

3.1 Relaxed Lazy Soundness

We pose relaxed lazy soundness as an intersection of relaxed and lazy soundness properties. Consequently, a model is relaxed lazy sound if and only if (i) in each reachable state, there is no

more than one token in o , (ii) it contains some feasible executions that end at $M \geq [o]$, and (iii) each model activity is present in at least one feasible execution. We formally define relaxed lazy soundness for DPNs as follows:

Definition 3. Let \mathcal{N} be a DPN with initial state (M_I, α_I) and distinguished sink place o . Let $\mathcal{M}_F = \{M_F | M_F \in \mathcal{M}_{\mathcal{N}} \wedge M_F(o) = 1 \wedge M_F \geq [o]\}$. \mathcal{N} is relaxed lazy sound if and only if the following properties hold:

- 1) for each $M \in \mathcal{M}_{\mathcal{N}}$, $M(o) \leq 1$;
- 2) for each $t \in T$, there exist $(M, \alpha), (M', \alpha') \in Reach_{\mathcal{N}}$ and $M_F \in \mathcal{M}_F$, such that $(M, \alpha)[t](M', \alpha')$ and $(M', \alpha')[*](M_F, \alpha_F)$ for some α_F .

The first condition states that place o should be bounded and contain no more than one token. The second condition verifies that each DPN transition may fire leading to a state from which the sink place is reachable (potentially, with some other tokens in the net). DPN \mathcal{N} from Fig. 1 is relaxed lazy sound, since both these conditions hold, although the DPN is unbounded and can enter a deadlock if a gambler always loses. Compared with lazy-only and relaxed-only soundness, relaxed lazy soundness can potentially be used for a wider range of resource-oriented models, since it captures the basic properties that should typically hold for such models.

4. Verification Algorithm

In this section, we introduce the algorithm to verify relaxed lazy soundness of a DPN. The algorithm is based on constructing and investigating a state space abstraction called an abstract coverability graph (ACG) that we define in the next subsection.

4.1 State Space Abstraction

An ACG of a DPN is a generalization of a classical coverability graph such that each node represents not a single state but a set of states having the same marking but different variable valuations. Compared with a classical coverability graph, the abstract version is always finite for the DPN setting that we consider in this paper (real-typed variables, variable-operator-constant/variable-operator-variable conditions).

To represent a set of variable valuations, we use the language of constraints defined in the previous section. In [22], we have proved that any set of variable valuations in DPN abstract state space structures can be described by some formula in $\Phi(V)$. The formula of a new state is then computed using operator \oplus defined in [18]. Given transition constraint $guard(t)$ and node formula $\varphi_n \in \Phi(V)$, $[[\varphi_n \oplus guard(t)]]$ is the union of all possible variable valuations that can be obtained by firing t at any variable valuation from $[[\varphi_n]]$. The result of this operation is computed using the concept of quantifier elimination, which is decidable for real arithmetic [25], but may be undecidable for other domains.

To define an ACG, we first define a coverability relation. Let $(M, \varphi), (M', \varphi')$ be two nodes. We say that (M', φ') covers (resp., strictly covers) (M, φ) , denoted as $(M, \varphi) \sqsubseteq (M', \varphi')$ (resp., $(M, \varphi) \sqsubset (M', \varphi')$), if and only if $[[\varphi]] = [[\varphi']]$ and $M \leq M'$ (resp., $M < M'$). To operate with unbounded nets, we use the special symbol ω , as in [22], which represents an unbounded number of tokens. For each integer n , $\omega > n$, $\omega \pm n = \omega$ and $\omega \geq \omega$. Now we can define an ACG:

Definition 4 (Abstract Coverability Graph). Let $\mathcal{N} = \langle P, T, F, V, guard \rangle$ be a DPN with initial state (M_I, α_I) . Let $\Phi(V)$ be the language of constraints, as in Section 3. Abstract Coverability Graph $ACG_{\mathcal{N}}$ of \mathcal{N} is a tuple $\langle S, E, s_0 \rangle$, where:

- $s_0 = (M_I, \phi_I) \in S$ is the initial node with $\phi_I = \bigwedge_{v \in V} \{v = \alpha_I(v)\}$;
- $S \subseteq \mathcal{M}_{\mathcal{N}} \times \Phi(V)$ is the least set that contains s_0 and is closed under the transition relation.
- $E \subseteq S \times T \times S$ is a set of arcs labeled with transitions, s.t. $((M, \phi), t, (M', \phi')) \in E$ if and only if:

- $\phi' = \phi \oplus guard(t)$ and $[[\phi]] \neq \emptyset$.
- for each $p \in P$, $M(p) \geq F(p, t)$.
- given $M^*(p) = M(p) - F(p, t) + F(t, p)$, $M'(p) = \omega$ if there exists a node $(M'', \phi'') \in S_{CG}$ along the path from s_0 to (M, ϕ) , s.t. $(M'', \phi'') \sqsubset (M^*, \phi')$, and $M^*(p) > M''(p)$, otherwise $M'(p) = M^*(p)$.

If $(s, t, s') \in E$, we say that (s, t, s') is a transition firing in $ACG_{\mathcal{N}}$. We denote a transition firing by writing $s[t]s'$. We extend this definition to sequences $\sigma = \langle t_1, \dots, t_n \rangle$ of n transition firings, called traces, and denote the corresponding run by $s_0[t_1]s_1[t_2] \dots [t_n]s_n$ or equivalently by $s_0[\sigma]s_n$. Fig. 3 illustrates an ACG constructed for DPN \mathcal{N} from Fig. 1.

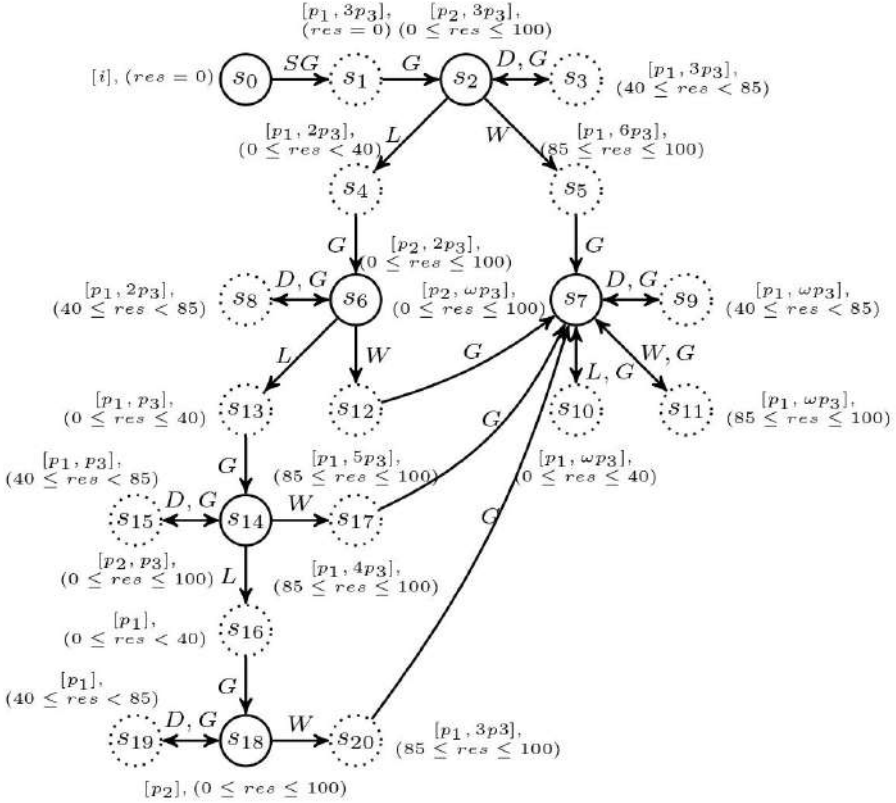


Fig. 3. An abstract coverability graph for \mathcal{N} from Fig. 1. Arcs are labeled with the initial letters of the transition names. Square brackets denote markings. Dotted circles denote nodes from which it is possible to fire End Gambling and reach o (nodes with a token in o are omitted for brevity).

Proposition 1. Let $\mathcal{N} = \langle P, T, F, V, guard \rangle$ be a DPN with initial state (M_I, α_I) and $RG_{\mathcal{N}}$ be a reachability graph of \mathcal{N} . Let $ACG_{\mathcal{N}}$ be an abstract coverability graph of \mathcal{N} . Then, $(M_I, \phi_I)[\sigma](M, \phi)$ is in $ACG_{\mathcal{N}}$ if for each $\alpha \in [[\phi]]$, $(M_I, \alpha_I)[\sigma](M^*, \alpha)$ is in $RG_{\mathcal{N}}$, where $M^*(p) = M(p)$ if $M(p) \neq \omega$.

Proof. The statement is a corollary to Definition 4 (specifically, from the consequent application of the \oplus operator and from the marking construction that is based on the coverability relation). \square

Proposition 2. Let $\mathcal{N} = \langle P, T, F, V, guard \rangle$ be a DPN with initial state (M_I, α_I) and $RG_{\mathcal{N}}$ be the reachability graph of \mathcal{N} . Let $ACG_{\mathcal{N}}$ be the abstract coverability graph of \mathcal{N} . If run $(M_I, \phi_I)[\sigma](M, \phi)$ is in $ACG_{\mathcal{N}}$, then for each $\alpha \in [[\phi]]$, there exists run $(M_I, \alpha_I)[\sigma^*](M^*, \alpha)$ in

$RG_{\mathcal{N}}$, where the set of transitions in σ^* coincides with the set of transitions in σ , and $M(p) = M^*(p)$ if $M(p) \neq \omega$.

Proof. If $\text{run}(M_I, \phi_I)[\sigma](M, \phi)$ is in $ACG_{\mathcal{N}}$ and for each p , $M(p) \neq \omega$, then for each $\alpha \in [[\phi]]$, $RG_{\mathcal{N}}$ includes $\text{run}(M_I, \alpha_I)[\sigma](M, \alpha)$ (follows from Definition 4). Assume that a run in $ACG_{\mathcal{N}}$ ends with (M, ϕ) , where for some p , $M(p) = \omega$. Then, trace σ may not be executable in $RG_{\mathcal{N}}$ (due to the nature of ω -abstraction). Let us represent σ as a sequence $\sigma_1, t_1, \dots, t_{n-1}, \sigma_n$, so that the execution of each σ -part in $ACG_{\mathcal{N}}$ preserves the same set of places with ω in all the markings, while transitions t_1, \dots, t_{n-1} transform some places to ω -places.

Consider a base case. Let $(M_I, \phi_I)[\sigma_1](M_1, \phi_1)$ be a run in $ACG_{\mathcal{N}}$. For brevity, we further denote (M, A) as a set of DPN states (M, α) , where $\alpha \in A$. Let (M_1, A_1) be a set of states resulting from executing σ_1 from (M_I, α_I) in $RG_{\mathcal{N}}$. By Definition 4, $A_1 = [[\phi_1]]$. Let $(M_1, \phi_1)[t_1](M'_1, \phi'_1)$ be a transition firing in $ACG_{\mathcal{N}}$. Then, there exists a sequence of transition firings in \mathcal{N} that includes only transitions from σ_1 and transition t_1 and that can be executed any number of times by adding tokens in places for which $M'_1(p) = \omega$. From each node (M_1, α_1) with $\alpha_1 \in A_1$, execute t and this sequence of transitions as many times as necessary to obtain the number of tokens in each $p \in P$, where $M'_1(p) = \omega$, greater than any fixed but arbitrary natural number. Following Definition 4, there must exist a set of runs that include only transitions from σ_1 and transition t_1 and reach (M_1^*, α_1') , where $M_1^*(p) = M'_1(p)$ if $M_1^*(p) \neq \omega$, so that the union of such (M_1^*, α_1') can be described as (M_1^*, A_1') with $A_1' = [[\phi_1']]$.

Induction step. Let for $k < n$, it is true. Then, we have $\text{run}(M_k', \phi_k')[\sigma_{k+1}](M_{k+1}, \phi_{k+1})$ in $ACG_{\mathcal{N}}$ and there exists a set of states (M_k^*, A_k') such that $A_k' = [[\phi_k']]$ and $M_k^*(p) = M_k'(p)$ if $M_k'(p) \neq \omega$. Since for each $p \in P$, where $M_k^*(p) = \omega$, we have a number of tokens greater than any fixed but arbitrary natural number, trace σ_{k+1} can be executed on (M_k^*, A_k') leading to the set of states (M_{k+1}^*, A_{k+1}) , where $A_{k+1} = [[\phi_{k+1}]]$ and $M_{k+1}^*(p) = M_{k+1}(p)$ if $M_{k+1}(p) \neq \omega$. If σ_{k+1} is the last part of σ , the proposition is proved. Otherwise, there must exist transition firing $(M_{k+1}, \phi_{k+1})[t_{k+1}](M_{k+1}', \phi_{k+1}')$ in $ACG_{\mathcal{N}}$. Then, there exists a sequence of transitions that consists of transitions from $\sigma_1, \dots, \sigma_{k+1}$, transitions t_1, \dots, t_{k+1} , that ends with t_{k+1} , and that can be executed any number of times putting tokens in places, for which $M_{k+1}'(p) = \omega$ and $M_{k+1}(p) \neq \omega$. From (M_{k+1}^*, A_{k+1}) , let execute t_{k+1} and this sequence of transitions as many times as necessary to obtain the number of tokens in each $p \in P$, where $M_{k+1}'(p) = \omega$ and $M_{k+1}(p) \neq \omega$, greater than any fixed but arbitrary natural number. After that, we obtain the set of states (M_{k+1}^*, A_{k+1}') , such that $A_{k+1}' = [[\phi_{k+1}']]$ and $M_{k+1}^*(p) = M_{k+1}'(p)$ if $M_{k+1}'(p) \neq \omega$. \square

Note that it is impossible to tell from a coverability graph alone whether the net contains deadlocks and/or livelocks. This was proved in [24] for classical coverability graphs, and since an ACG is a generalization of this state-transition structure, the same limitation holds for ACG. For bounded DPNs, the problem of detecting deadlocks and livelocks is known to be decidable and was solved in [18,22,23]. For unbounded DPNs, there is currently no solution for this task. Note that the task of detecting deadlocks is challenging even for classical Petri nets. Although deadlock- and livelock-freedom can be reduced to the reachability problem that is known to be decidable [11], existing works, such as [24-25], allow one to verify the absence of deadlocks only for rather limited classes of nets (e.g., [25] considers nets with only one unbounded place). For similar reasons, the coverability graphs alone cannot be used to check whether the sink place is reachable in a 'clean way' (when $M = [o]$). Since coverability graphs allow ω -markings, we cannot determine whether it is possible to consume all the tokens from ω -places and thus reach the output place with no remaining tokens. There are some workarounds for this, but most of them use other representations of business processes, such as π -calculus in [28].

Nevertheless, coverability graphs could be used to verify some important properties that should usually hold for resource-oriented models. It is possible to check whether the model could terminate having one token in o and potentially extra tokens in the net (we could call this as 'adequate' termination), whether each model transition can actually fire and be present in executions with

‘adequate’ termination, and whether all the model executions do not put more than one token in σ . These properties could be seen as a combination of main relaxed and lazy soundness features, and all of them are actually included in the proposed above the relaxed lazy soundness property.

4.2 Checking Relaxed Lazy Soundness

The procedure of checking relaxed lazy soundness is based on constructing and studying the ACG. Algorithm 1 illustrates the procedure of constructing an ACG according to Definition 4. Sets S and E represent the nodes and arcs of $ACG_{\mathcal{N}}$, s_0 – the initial node of $ACG_{\mathcal{N}}$. Set N stores the nodes that need to be expanded. For each node in \mathcal{N} , we try to fire DPN transitions for the marking and variable valuations it represents (lines 6-11). If some transition may fire, we try to find preceding strictly covering nodes (lines 12-13). If such nodes are found, we define places where the number of tokens should be replaced with ω (lines 13-16). If the resulting node already exists in the graph, we add an arc to this node from the current one; otherwise, a new node is added to the graph and to the set of nodes to be expanded (lines 17-20).

Regarding the implementation of the algorithm, the simplest approach is to construct the graph using a depth-first search and use a stack to represent set N . For each new node in the graph, we can determine the set of parents using the information obtained from the previous node, and thus skip the graph traversal procedure when checking node coverage. As for checking the equivalence of formulas, we can perform it by checking the satisfiability: to check $\phi \sim \phi'$, we can verify whether $\phi \wedge \neg\phi' \vee \neg\phi \wedge \phi'$ is satisfiable (if not, then $\phi \sim \phi'$ is true) using known satisfiability modulo theories (SMT) solvers such as Z3 [29]. Note that for our constraint language, satisfiability is decidable, as well as the quantifier elimination needed to compute the result of the \oplus -procedure [22].

Algorithm 1 *ConstructACG*($\mathcal{N}, (M_I, \alpha_I)$)

Input: A DPN $\mathcal{N} = \langle P, T, F, V, guard \rangle$ with initial state (M_I, α_I) .

Result: Abstract Coverability Graph of \mathcal{N} .

$\phi_I \leftarrow \bigwedge_{v \in V} \{v = \alpha_I(v)\}$

$s_0 \leftarrow (M_I, \alpha_I)$

$S \leftarrow \{s_0\}$

$E \leftarrow \emptyset$

$N \leftarrow \{s_0\}$

while $N \neq \emptyset$ **do**

$(M, \phi) \leftarrow Pick(N)$ // Take a node from N

$N \leftarrow N \setminus \{(M, \phi)\}$

foreach $t \in T$ s.t. $M[t]M'$ **do**

$\phi' \leftarrow \phi \oplus guard(t)$

if $\neg(\phi' \sim false)$ then

foreach state (M_o, ϕ_o) , from which (M, ϕ) is reachable **do**

if $(M' > M_o) \wedge (\phi' \sim \phi_o)$ then

foreach $p \in P$ **do**

if $M'(p) > M_o(p)$ then

$M'(p) = \omega$

$E \leftarrow E \cup \{(M, \phi), t, (M', \phi')\}$

if $\forall (\bar{M}, \bar{\phi}) \in S: \bar{M} \neq M' \vee \neg(\phi' \sim \bar{\phi})$ then

$S \leftarrow S \cup \{(M', \phi')\}$

$N \leftarrow N \cup \{(M', \phi')\}$

return $\langle S, E, s_0 \rangle$

Algorithm 2 describes the procedure for checking relaxed lazy soundness that consists of two steps:

- (i) constructing an ACG (line 1),
- (ii) exploring the ACG using graph-traversing techniques to verify the absence of markings that have more than one token in o (lines 2-3) and the ability to reach the sink place after each of the DPN transitions (lines 4-7).

Some of these checks can be included in the ACG construction procedure to answer the soundness of the model more quickly.

Algorithm 2 *CheckRelaxedLazySoundness*($\mathcal{N}, (M_I, \alpha_I)$)

Input: A DPN $\mathcal{N} = \langle P, T, F, V, guard \rangle$ with initial state (M_I, α_I) .

Result: Whether or not \mathcal{N} is relaxed lazy sound.

$\{S, E, s_0\} \leftarrow \text{ConstructACG}(\mathcal{N}, M_I, \alpha_I)$

if $\exists (M, \phi) \in S: M(o) > 1$ then

return false

$S_{feasible} \leftarrow \{s \in S \mid \exists (M_F, \phi): s[*](M_F, \phi) \wedge M_F > [o]\}$

foreach $t \in T$ **do**

if $\forall (s, t, s'): s' \notin S_{feasible}$ **then**

return false

return true

Proposition 3. Let $\mathcal{N} = \langle P, T, F, V, guard \rangle$ be a DPN with initial state (M_I, α_I) . Procedure *CheckRelaxedLazySoundness*($\mathcal{N}, (M_I, \alpha_I)$) terminates.

Proof. In paper [22], it was proved that a labeled transition system for a DPN with quasi-ordering \sqsubseteq defined there is a well-structured transition system (WSTS) [30]. The ACG defined in our paper is actually a coverability graph for a labeled transition system defined in [22], where sets of states are replaced with formulas of the constraint language. In our case, relation \sqsubseteq is decidable, and the procedure of computing node successors is also decidable (follows from the decidability of \sqsubseteq). From this, according to the WSTSs theory, it follows that the ACG is finite and effectively constructible. Since the ACG is finite, graph traversal methods on this structure are guaranteed to terminate. This proves that the procedure *CheckRelaxedLazySoundness* terminates. \square

Proposition 4. Let $\mathcal{N} = \langle P, T, F, V, guard \rangle$ be a DPN with initial state (M_I, α_I) . Let $ACG_{\mathcal{N}} = \langle S, E, s_0 \rangle$ be the abstract coverability graph of \mathcal{N} . Then for each $M \in \mathcal{M}_{\mathcal{N}}$, $M(o) \leq 1$ if and only if $M(o) \leq 1$ for each $(M, \phi) \in S$.

Proof. (\Rightarrow) It follows from Proposition 1 that in $ACG_{\mathcal{N}}$, $M(o)$ may be either 0, 1, or ω . If $M(o) = \omega$, then o is unbounded and $RG_{\mathcal{N}}$ contains an execution that produces M' with $M'(o) > 1$, which contradicts the precondition $\forall M \in \mathcal{M}_{\mathcal{N}}, M(o) \leq 1$.

(\Leftarrow) Follows from Proposition 2. \square

Proposition 5. Let $\mathcal{N} = \langle P, T, F, V, guard \rangle$ be a DPN with initial state (M_I, α_I) . Let $ACG_{\mathcal{N}} = \langle S, E, s_0 \rangle$ be the abstract coverability graph of \mathcal{N} . Then for each $t \in T$, $RG_{\mathcal{N}}$ contains an arc (v, t, v') included in a path to some node with $M_F \geq [o]$ if and only if there exists an arc $(s, t, s') \in E$ with a path from s' to a node with $M_F^\omega \geq [o]$.

Proof. (\Rightarrow) $RG_{\mathcal{N}}$ contains an execution that includes t and produces $M_F \geq [o]$. According to Proposition 1, the trace that represents this execution must be in $ACG_{\mathcal{N}}$. The trace execution ends at (M', ϕ') , where $M' \geq M_F$ (follows from Definition 4). Since $M' \geq M_F$, we have $M' \geq [o]$.

(\Leftarrow) $ACG_{\mathcal{N}}$ contains an execution that includes t and produces $M_F^\omega \geq [o]$. By Proposition 2, $RG_{\mathcal{N}}$ must exist a corresponding trace σ^* that includes all transitions of the trace in $ACG_{\mathcal{N}}$. Let this trace execution end at (M', α') . If $M_F^\omega(o) < \omega$, then $M'(o) = M_F^\omega(o) \geq 1$. If $M_F^\omega(o) = \omega$, then $M'(o)$ contains at least one token (follows from Definition 4). Thus, $M' \geq [o]$. \square

The following is a corollary of Propositions 4 and 5:

Corollary 1. Let $\mathcal{N} = \langle P, T, F, V, guard \rangle$ be some DPN with initial state (M_I, α_I) . \mathcal{N} is relaxed lazy soundif and only if *CheckRelaxedLazySoundness*($\mathcal{N}, (M_I, \alpha_I)$) returns *true*.

5. Implementation and Experiments

The proposed algorithm has been implemented as a module in the existing DPN soundness verification toolkit implemented on .NET WPF. The application with the added module is available for download on <https://github.com/SuvorovNM/DPN-Soundness-Verification>. Fig. 4 demonstrates the outputs provided by the application for DPN \mathcal{N} from Fig. 1. A custom coloring function is implemented to ease the interpretation of ACGs, which can be huge in some cases.

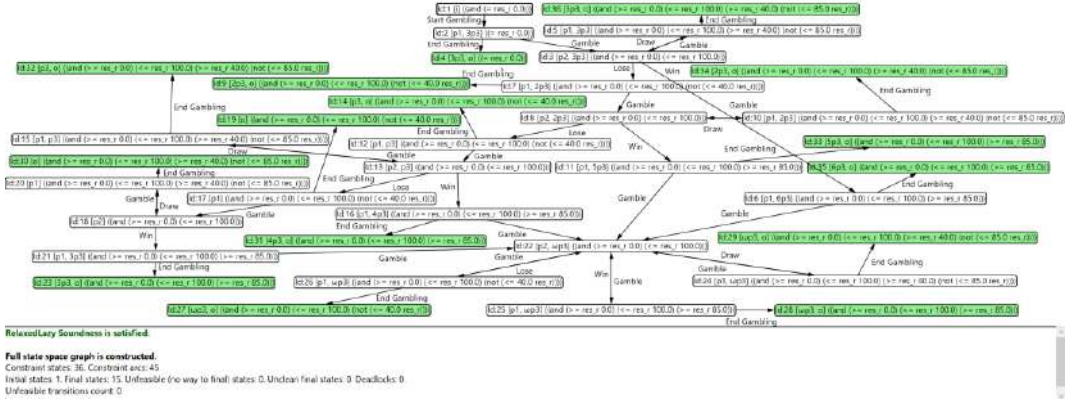


Fig. 4. Screenshot of the tool that implements the relaxed lazy soundness verification algorithm. The tool demonstrates $ACG_{\mathcal{N}}$ for DPN \mathcal{N} from Fig. 1.

The tool takes as input a DPN in an extended PNML-based file format. All operations over formulas are conducted with the help of Z3 Solver [29]. At the implementation level, we have also made a small adjustment to the algorithm: we decided to check for $M(o) \leq 1$ when constructing an ACG and immediately return false when a node with $M(o) > 1$ is obtained. For large models, the ACG can be huge: even for a DPN with 10 transitions, an ACG may have more than 100000 nodes (see the HugeACG.pnmlx example in the repository). Thus, building the entire ACG when it is already known to be unsound may not be reasonable in some situations.

Table 1. Relaxed lazy soundness verification time for sample DPNs. Column ‘Size’ defines the net size in the number of places, denoted P , the number of transitions, denoted T , and the number of variables, denoted V .

Model	Size	Classic Sound	Relaxed Lazy Sound	Verification Time: Classic	Verification Time: Relaxed Lazy
Gambling Example (Fig. 1)	5P + 6T + 1V	False	True	34ms	89ms
Livelock Example [22]	3P+3T+2V	True	True	130ms	18ms
Digital Whiteboard: Transfer [9]	7P+6T+3V	False	True	40ms	15ms
Package Handling [23]	16P+28T+5V	False	False	1010ms	213ms
Road Fines Mined [9]	9P+19T+8V	False	False	491ms	194ms
Simple Auction [31]	4P+4T+2V	False	True	193ms	90ms

Table 1 reports on how much time the proposed algorithm takes to verify the relaxed lazy soundness of different DPN models presented in the literature. For bounded DPNs, as expected, verifying relaxed lazy soundness is usually faster than classical soundness, since classical soundness requires constructing more refined state space structures. If a net is unbounded (as \mathcal{N} in Fig. 1), then classical soundness can be checked much faster, since we can immediately return *false* when a node with ω is obtained.

We have also tested the performance of the algorithm on synthetically generated data. Given $n \in \mathbb{N}$, we considered DPNs parameterized according to the following setup:

- $1.2n$ places,
- n transitions,
- $0.25n$ variables, and
- $0.5n$ conditions.

For each $n \in \mathbb{N}$ from 3 to 90, we generated 10 DPNs that have at least one trace leading to $M \geq [o]$ and at least 40% feasible transitions (i.e., transitions occurring in executions leading to some $M \geq [o]$) using the tool introduced in [22], refined for the current study. In our experiments, we have considered nets with a low and moderate level of inherent concurrency (a limitation of the DPN generation tool). The obtained results are visualized in Fig. 5. The plot shows that our algorithm typically takes less than a minute to verify the relaxed lazy soundness of a DPN with fewer than 100 transitions.

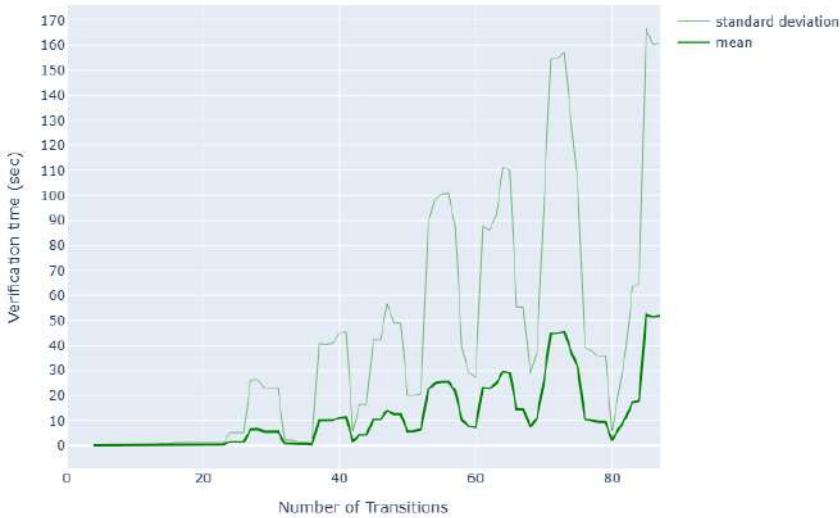


Fig. 5. *CheckRelaxedLazySoundnes* procedure execution time on DPNs of different sizes.

Note that in the worst-case scenarios, the verification can take an unreasonable amount of time. This may be the case for DPNs with a high inherent level of concurrency, which can lead to state space explosion, and for DPNs with long constraints on transitions, which can seriously impact the speed of formula operations. However, the results of our preliminary experimental evaluation show that the algorithm is promising and can be used in practice to verify the correctness of resource-oriented data-aware process models. Interestingly, the algorithm could also be used for preliminary verification of arbitrary process models, since for bounded models, it terminates sufficiently faster than the algorithm for classical soundness verification.

6. Conclusion

In this paper, we have proposed the notion of relaxed lazy soundness that can be used to check correctness of resource-oriented process models. We have shown that this property is decidable both for classical Petri nets and for Data Petri nets and could be evaluated using a single coverability graph construction compared to separate relaxed and lazy soundness properties that require solving a reachability problem to be decided. Performance evaluation of the algorithm implementation shows its practical applicability for process models of small and medium size.

The proposed algorithm can be used immediately after building a process model to verify that the model actually contains executions that terminate 'adequately' and that each process activity can be present in such executions. One of practical applications of the proposed algorithm could be a plug-in for tools for constructing data-aware process models. The algorithm could run in the background and warn if the user's model is ill-structured.

In the future, we plan to estimate our verification algorithm on real industrial process models both in terms of speed and in terms of model errors found. This could help to identify the overall practical applicability of the relaxed lazy soundness property in industry. Another direction for investigation is finding an abstract state space structure that would be smaller than the proposed ACG but that would be sufficient for verifying relaxed lazy soundness. In its current state, the algorithm provides rather poor results for large models with a high level of concurrency, and it is interesting whether this issue can be addressed. Lastly, a promising direction could be to define the rules to reduce an ACG in order to make its visualization perceivable even for models of medium and large sizes, where an ACG could contain hundreds or thousands of nodes.

References

- [1]. Weske M. *Business Process Management: Concepts, Languages, Architectures*. Introduction. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 3–23.
- [2]. van der Aalst W. M. P. Verification of workflow nets. In *Application and Theory of Petri Nets 1997*, 1997, pp. 407–426.
- [3]. Martens A. Analyzing web service based business processes. In *Fundamental Approaches to Software Engineering*, 2005, pp. 19–33.
- [4]. van Hee K. M., Sidorova N., Voorhoeve M. Soundness and separability of workflow nets in the stepwise refinement approach. In *Applications and Theory of Petri Nets*, 2003, pp. 337–356.
- [5]. Dehnert J., Rittgen P. Relaxed soundness of business processes. In *Advanced Information Systems Engineering*, 2001, pp. 157–170.
- [6]. Puhlmann F., Weske M. Investigations on soundness regarding lazy activities. In *Business Process Management*, 2006, pp. 145–160.
- [7]. van der Aalst W. M. P., van Hee K. M., ter Hofstede A. H. M., Sidorova N., Verbeek H. M. W., Voorhoeve M., Wynn M. T. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3), 2011, pp. 333–363.
- [8]. de Leoni M., van der Aalst W. M. P. Data-aware process mining: Discovering decisions in processes using alignments. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013, pp. 1454–1461.
- [9]. Mannhardt F. Multi-perspective process mining. Ph.D. dissertation, Eindhoven University of Technology, 2018.
- [10]. de Leoni M., Felli P., Montali M. A holistic approach for soundness verification of decision-aware process models. In *Conceptual Modeling*, 2018, pp. 219–235.
- [11]. Esparza J., Nielsen M. Decidability issues for Petri Nets. *Information Processing and Cybernetics*, 30(3), 1994, pp. 143-160.
- [12]. Czerwinski W., Lasota S., Lazić R., Leroux J., Mazowiecki F. The reachability problem for Petri nets is not elementary. In *STOC 2019*, 2019, pp. 24–33.
- [13]. Bazhenova E., Zerbato F., Oliboni B., Weske M. From bpmn process models to dmn decision models. *Information Systems*, vol. 83, 2019, pp. 69–88.
- [14]. Knuplesch D., Ly L. T., Rinderle-Ma S., Pfeifer H., Dadam P. On enabling data-aware compliance checking of business process models. In *Conceptual Modeling*, 2010, pp. 332–346.
- [15]. van der Aalst W. M. The application of petri nets to workflow management. *Journal of Circuits, Systems and Computers*, vol. 8, 1998, pp. 21–66.
- [16]. Awad A., Decker G., Lohmann N. Diagnosing and repairing data anomalies in process models. In *Business Process Management Workshops*, 2010, pp. 5–16.
- [17]. Sidorova N., Stahl C., Trcka N. Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. *Information Systems*, 36(7) , 2011, pp. 1026–1043.
- [18]. Felli P., de Leoni M., Montali M. Soundness verification of decision-aware process models with variable-to-variable conditions. In *ACSD 2019*, 2019, pp. 82–91.

- [19]. Tao X., Liu G., Yang B., Yan C., Jiang C. Workflow nets with tables and their soundness. *IEEE Transactions on Industrial Informatics*, 16(3), 2020, pp. 1503–1515.
- [20]. Song J., Liu G, Wang M. Model checking of workflow nets with tables and constraints. *ACM Transactions on Autonomous and Adaptive Systems*, 2025, 37p.
- [21]. Felli P., Montali M., Winkler S. Soundness of data-aware processes with arithmetic conditions. In *Advanced Information Systems Engineering*, 2022, pp. 389–406.
- [22]. Suvorov N. M., Lomazova I. A. Verification of data-aware process models: Checking soundness of data petri nets. *Journal of Logical and Algebraic Methods in Programming*, vol. 138, 100953, 2024.
- [23]. Felli P., de Leoni M., Montali M. Soundness verification of data-aware process models with variable-to-variable conditions. *Fundamenta Informaticae*, 182(1), 2021, pp. 1–29.
- [24]. Murata T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4), pp. 541–580, 1989.
- [25]. Tarski A. A decision method for elementary algebra and geometry. *Journal of Symbolic Logic*, 14(3), pp. 188–188, 1949.
- [26]. Lu F., Tao R., Du Y., Zeng Q., Bao Y. Deadlock detection-oriented unfolding of unbounded petri nets. *Information Sciences*, vol. 497, 2019, pp. 1–22.
- [27]. Ding Z., Jiang C., Zhou M. Deadlock checking for one-place unbounded petri nets based on modified reachability trees. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(3), pp. 881–883, 2008.
- [28]. Puhlmann F. Soundness verification of business processes specified in the pi-calculus. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, 2007, pp. 6–23.
- [29]. de Moura L., Björner N. Z3: An efficient smt solver. In *TACAS*, 2008, pp. 337–340.
- [30]. Finkel A., Schnoebelen P. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1), 2001, pp. 63–92.
- [31]. Felli P., Montali M., Winkler S. Repairing soundness properties in data-aware processes. In *2023 5th International Conference on Process Mining (ICPM)*, 2023, pp. 41–48.

Информация об авторах / Information about authors

Николай Михайлович СУВОРОВ – аспирант аспирантской школы по компьютерным наукам НИУ ВШЭ, стажер-исследователь лаборатории процессно-ориентированных информационных систем (ПОИС) факультета компьютерных наук НИУ ВШЭ. Сфера научных интересов: теория автоматов, распределенные процессы, верификация и исправление моделей.

Nikolai Mikhailovich SUVOROV – postgraduate student at Doctoral School of Computer Science, HSE University, research fellow at Laboratory of Process-Aware Information Systems, Faculty of Computer Science, HSE University. Research interests: automata theory, distributed processes, and model verification and repair.

Ирина Александровна ЛОМАЗОВА – доктор физико-математических наук, профессор факультета компьютерных наук НИУ ВШЭ, научный руководитель научно-учебной лаборатории процессно-ориентированных информационных систем (ПОИС) НИУ ВШЭ. Область научных интересов: анализ и моделирование бизнес-процессов, сети Петри, вложенные сети Петри, процессно-ориентированные информационные системы, формальные модели распределённых систем.

Irina Alexandrovna LOMAZOVA – Dr. Sci. (Phys.-Math.), professor of the faculty of computer science in HSE University, and laboratory head of the Laboratory for Process-Aware Information Systems (PAIS Lab), HSE University. Doctor of Sciences in Theoretical Foundations of Computer Science Russian Academy of Sciences Dorodnitsyn Computation Center since 2002. Research interests mainly include analysis and modeling of business processes, Petri nets, process-oriented information systems, formal models of distributed systems.

DOI: 10.15514/ISPRAS-2025-37(4)-20



Выбор ракурсов с помощью обратного рендеринга геометрических моделей для реконструкции материалов

¹ В.В. Санжаров, ORCID: 0000-0001-6455-6444 <vadim.sanzharov@graphics.cs.msu.ru>

^{1,2} В.А. Фролов, ORCID: 0000-0001-8829-9884 <vladimir.frolov@graphics.cs.msu.ru>

² В.А. Галактионов, ORCID: 0000-0001-6460-7539 <vlgal@gin.keldysh.ru>

¹ Московский государственный университет имени М.В. Ломоносова,
Россия, 119991, Москва, Ленинские горы, д. 1.

² Институт прикладной математики им. М.В. Келдыша РАН,
Россия, 125047, Москва, Миусская пл., д. 4.

Аннотация. Методы выбора точки обзора для 3D сцен находят применение в компьютерном зрении, компьютерной графике и научной визуализации для того, чтобы получить ракурсы, наиболее подходящие для решаемой задачи. В данной работе предлагается метод для выбора точек обзора на основе обратного рендеринга в приложении к задаче реконструкции материалов. Предложенный метод решает задачу выбора произвольных ракурсов (то есть не из предопределенного набора) на основе различных оценок качества ракурсов, использующих геометрические характеристики целевого 3D объекта. Предложенный метод позволяет использовать реализацию обратного рендеринга как на основе дифференцируемого рендеринга, так и с помощью безградиентных методов оптимизации. Предложенный метод был протестирован на открытом наборе данных для 3D реконструкции. Тестирование показало прирост в качестве реконструкции при использовании предложенного метода с различными методами оценки качества ракурсов по сравнению с наивными стратегиями выбора точек обзора.

Ключевые слова: обратный рендеринг; реконструкция материалов; выбор ракурсов.

Для цитирования: Санжаров В.В., Фролов В.А., Галактионов В.А. Выбор ракурсов с помощью обратного рендеринга геометрических моделей для реконструкции материалов. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 85–102. DOI: 10.15514/ISPRAS–2025–37(4)–20.

Viewpoint Selection for Material Reconstruction Using Inverse Rendering of Geometric Models

¹ V.V. Sanzharov, ORCID: 0000-0001-6455-6444 <vadim.sanzharov@graphics.cs.msu.ru>

^{1,2} V.A. Frolov, ORCID: 0000-0001-8829-9884 <vladimir.frolov@graphics.cs.msu.ru>

² V.A. Galaktionov, ORCID: 0000-0001-6460-7539 <vlgal@gin.keldysh.ru>

¹ *Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia.*

² *Keldysh Institute of Applied Mathematics RAS,
Miusskaya sq., 4, Moscow, 125047, Russia*

Abstract. Viewpoint selection methods for 3D scenes are used in computer vision, computer graphics and scientific visualization to obtain views that are most suitable for the problem at hand. In this paper, a method for viewpoint selection based on inverse rendering is proposed for material reconstruction. The proposed method solves the problem of selecting arbitrary views (i.e., not from a predefined set) based on various view quality estimates using geometric characteristics of the target 3D object. The proposed method allows using both differentiable rendering-based and gradient-free optimization implementations of inverse rendering. The proposed method was tested on an open dataset for 3D reconstruction. Testing showed an increase in reconstruction quality when using the proposed method with various view quality estimates compared to naive viewpoint selection strategies.

Keywords: inverse rendering; materials reconstruction; viewpoint selection.

For citation: Sanzharov V.V., Frolov V.A., Galaktionov V.A. Viewpoint selection for material reconstruction using inverse rendering of geometric models. *Trudy ISP RAN/Proc. ISP RAS*, vol. 37, issue 4, part 2, 2025, pp. 85-102 (in Russian). DOI: 10.15514/ISPRAS-2025-37(4)-20.

1. Введение

Суть задачи реконструкции трехмерной сцены заключается в создании её описания в определённом формате на основе набора изображений реальной сцены. В случае представления описания 3D-сцены в «классическом» для компьютерной графики виде (полигональные сетки для геометрии, текстуры и скалярные значения для материалов и освещения) задачу можно разделить на реконструкцию её геометрического представления и реконструкцию информации об освещении. При использовании же полей освещённости (например, NeRF – neural radiance fields, нейронные поля освещённости) [1] или «gaussian splats» (GS) [2] обычно рассматривается совместное восстановление геометрии и освещения. Однако, такие представления сцены очень разнообразны и их трудно совмещать друг с другом, а также с классическими представлениями. Кроме того, такие решения обычно потребляют значительные объёмы памяти и вычислений при прямом рендеринге, что требует отдельного этапа оптимизации представления [3-4]. Существуют решения, которые одновременно восстанавливают поля освещённости, полигональные модели и модели материалов [5], однако их обучение требует значительных вычислительных ресурсов (авторы пишут о двух неделях обучения на 64-х H100 GPU). Поэтому, несмотря на бурное развитие различных представлений 3D сцен, классические модели компьютерной графики сохраняют свою актуальность. Следует отметить, что раздельная реконструкция геометрии и освещения с помощью обратного рендеринга также демонстрирует более высокое качество реконструкции [6]. Для работы с ними в задаче реконструкции используют методы обратного рендеринга, которые по своей сути решают задачу оптимизации параметров 3D сцены, чтобы изображение, полученное при прямом рендеринге, максимально совпадало с эталонной фотографией (рис. 1).

Наибольшее распространение получили методы обратного рендеринга, основанные на градиентной оптимизации [7-8] и использовании автоматического дифференцирования. В

частности, метод «reverse mode», так как в обратном рендеринге обычно имеется большое количество входных значений (например, при реконструкции материалов – пиксели текстур, задающих оптические свойства поверхности) и одно выходное значение функции потерь. В этом смысле задача обратного дифференцируемого рендеринга похожа на задачу обучения нейросетевых моделей. Однако, алгоритм трассировки путей, являющийся основой фотореалистичного рендеринга, при применении обратного дифференцирования дает граф вычислений высокой сложности, что привело к необходимости создания специальных методов [9] для решения проблем затрат памяти и вычислений.

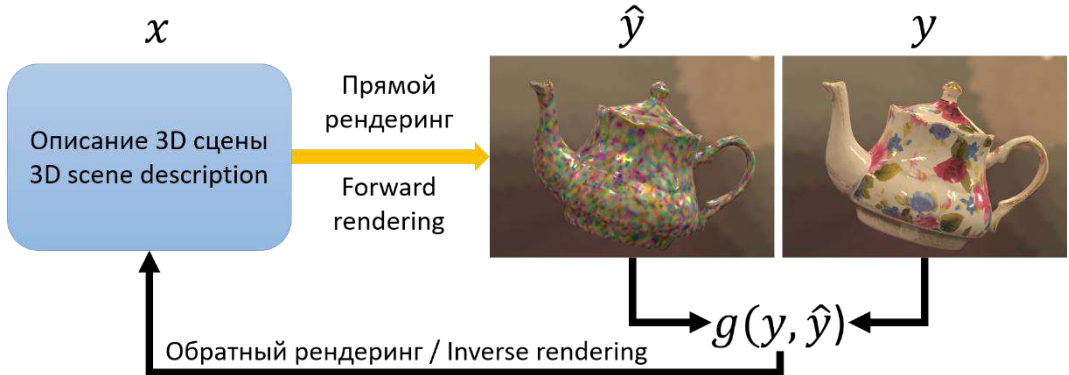


Рис. 1. Схема работы обратного рендеринга.
Fig. 1. Inverse rendering workflow.

Тем не менее, задача обратного рендеринга остается достаточно вычислительно затратной. При этом время реконструкции напрямую зависит от числа эталонных изображений, подаваемых на вход [10].

Чтобы точно восстановить текстуры оптических свойств поверхности, изображения на входе должны демонстрировать целевой объект реконструкции со всех сторон. Другими словами, если геометрическая модель уже известна, каждой полигон объекта должен быть виден хотя бы на одном ракурсе (если это возможно). Существующие наборы данных для оценки методов реконструкции, такие как [6, 11-12], обычно представляют собой изображения целевого объекта с ракурсов, полученных с помощью поворотного столика или путем перемещения камеры вокруг него. Это, с одной стороны, не гарантирует достаточную видимость всех частей объекта, а с другой обычно приводит к тому, что соседние изображения могут в значительной мере перекрываться и, вероятно, вносить небольшой вклад в точность реконструкции, увеличивая при этом время расчетов. В то же время существующие исследования показывают, что разнообразие выбранных точек обзора влияет на конечное качество реконструкции [13], улучшая результат по метрике PSNR до 1.9 дБ по сравнению со случайным выбором.

В данной работе рассматривается задача выбора набора ракурсов целевого объекта для улучшения качества реконструкции материалов и снижения вычислительных затрат.

2. Существующие решения

Выбор ракурса является одной из основных задач активного восприятия (active perception). Данная задача заключается в том, что необходимо выбрать положение камеры, которое максимально улучшит понимание сцены. В компьютерном зрении и графике эффективный выбор ракурсов может значительно снизить неопределенность в 3D-реконструкции или в синтезе новых ракурсов. Были предложены различные критерии, от основанных на теории информации (например, энтропии Шеннона) до покрытия геометрического представления и эвристик, основанных на неопределенности.

Существует достаточно много методов выбора ракурсов. В работах [14-15] представлен обзор различных метрик, классифицированных по типу используемых данных: площадь полигонов, силуэт, глубина, кривизна, семантическая информация, салиентность. Наибольшее распространение получили методы, основанные на теории информации, такие как расчет энтропии [16-18] и взаимной информации [19] с использованием геометрических данных. Также существуют подходы на основе глубокого обучения [10, 20].

В работе [10] предлагается выбор ракурсов на основе площади полигонов в текстурном пространстве. Этот и другие критерии применяются авторами для выбора изображений из заранее подготовленного набора для реконструкции текстур. Также предлагается алгоритм последовательного выбора нескольких ракурсов так, чтобы каждый полигон объекта был виден хотя бы на одном из ракурсов.

Основное ограничение этих методов связано с зависимостью от predetermined набора изображений. Большинство решений работают с заранее заданным набором изображений объекта, полученных путем рендеринга 3D-модели с фиксированных позиций камеры. Эти позиции часто определяются точками на сфере вокруг объекта [16-19] или вершинами икосаэдра [21], который затем тесселируется для аппроксимации сферы.

В работе [22] авторы расширяют NeRF-представление, рассматривая освещенность в каждой точке не как одно значение, а как распределение Гаусса, дисперсия которого описывает неопределенность. Новые положения камеры выбираются так, чтобы максимизировать уменьшение неопределенности модели. Это позволяет улучшить качество при построении поля освещенности по небольшому числу изображений, выбирая наиболее информативные.

В [23] предлагается построение предварительного поля освещенности для расчета глобальной карты неопределенности. Далее производится выбор ракурсов на основе этой карты. Другие подходы на основе нейронного рендеринга действуют схожим образом, – производится обучение нейронной модели для небольшого числа изображений, которая затем используется для оценки неопределенности с помощью различных эвристик. На основе полученных оценок выбираются новые ракурсы. Основной недостаток этих решений в том, что процесс выбора ракурсов в них требует значительных вычислительных затрат, связанных с постоянным обучением или дообучением нейронных моделей.

Работы [24-25] строят модели, позволяющие сразу получить очередной оптимальный ракурс из текущего. Однако, выбор осуществляется из predetermined набора ракурсов. Кроме того, для обучения моделей требуются эталонные оптимальные ракурсы.

В работе [26] авторы обучают нейросетевую модель, которая предсказывает карту неопределенности по одному изображению, что позволяет избежать построения специального поля освещенности.

Задача выбора ракурсов также использовалась и для «gaussian splatting»-представлений сцен. В [27] оценивается плотность сцены, которая затем используется для выбора ракурсов на основе максимального пространственного заполнения. В [28-29] используется плотность и глубина, чтобы оценить число вокселей, которые не видимы на данном ракурсе, но будут видимы на ракурсе-кандидате.

В целом, выбор ракурсов в приложении к полям освещенности, gaussian splatting и близким к ним представлениям основан на оценке неопределенности и активном обучении. Оценка неопределенности обычно строится на основе дисперсии цвета и/или плотности. Используемые стратегии выбора ракурса включают [13]: случайный выбор, выбор ракурса, наименее похожего на текущий и выбор на основе прироста информации (уменьшении неопределенности).

Таким образом, методы выбора ракурсов для классических представлений сцены с разделением геометрии (полигональных моделей) и моделей материалов опираются на выбор из predetermined набора ракурсов, который обычно генерируется по сетке на сфере/полусфере вокруг объекта. Это приводит к тому, что возможные оптимальные ракурсы

могут быть пропущены, а также к ограниченности позиционирования камеры – обычно допускается только вращение вокруг объекта на фиксированном расстоянии.

Большинство методов, ориентированных на нейронные представления, лишено этого недостатка, но приобретают другие. В частности, они требуют значительных вычислительных затрат на выбор ракурса, связанных либо с необходимостью (до)обучения вспомогательного нейронного представления (зачастую после выбора каждого нового ракурса), либо с обучением отдельной нейросетевой модели оценке неопределенности [26]. Также некоторые методы требуют наборов данных с эталонными оптимальными ракурсами. Данная работа посвящена методу выбора произвольных ракурсов (без использования предопределенного набора) для классических представлений сцены. Мы рассматриваем задачу реконструкции материалов, предполагая, что реконструкция геометрической модели в виде полигональной сетки уже некоторым образом выполнена.

3. Предлагаемое решение

Предлагаемый метод использует обратный рендеринг, чтобы подобрать набор положений виртуальной камеры, основываясь на некоторой оценке качества ракурса, вычисленной одним из способов, использующих геометрические данные [10, 14-15]. Далее рассмотрим схему работы предлагаемого метода, а конкретные использованные в работе оценки качества ракурса будут рассмотрены позднее.

3.1 Общая схема работы

Общая схема работы предлагаемого метода представлена на рис. 2.

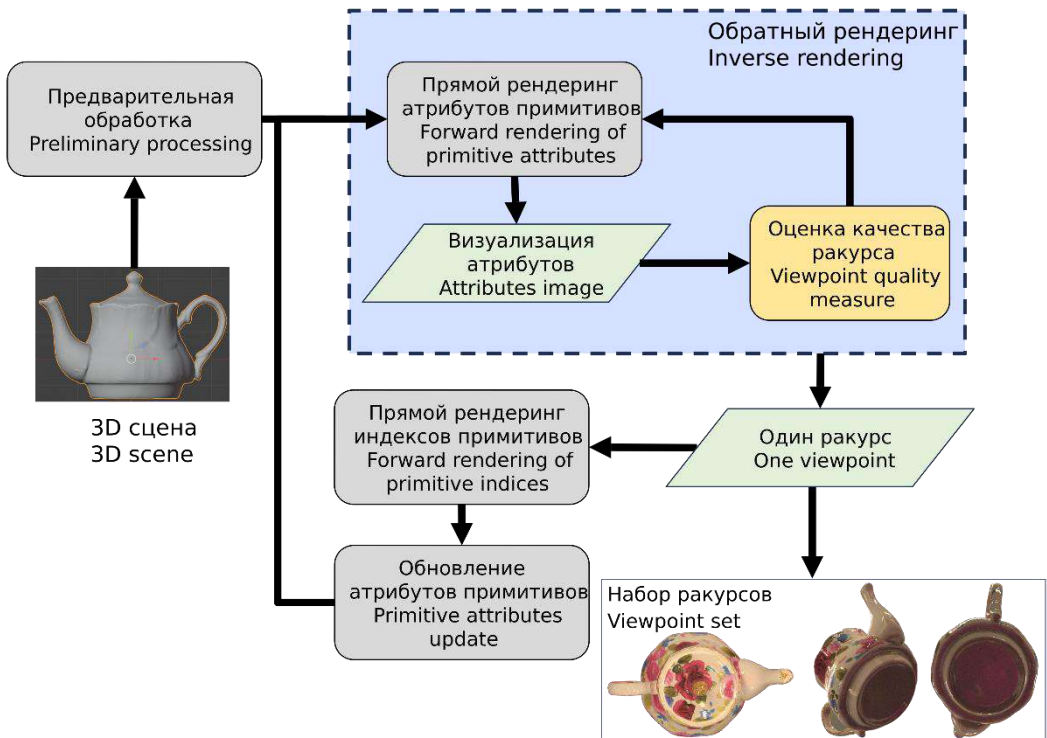


Рис. 2. Схема работы предлагаемого решения.
Fig. 2. Proposed solution schematic workflow.

На вход методу подается 3D-сцена, включающая один целевой 3D-объект, размещенный в начале координат. Использование только одного объекта – наиболее распространенный сценарий в существующих исследованиях и наборах данных [6, 11, 12]. При необходимости обрабатывать нескольких объектов сразу, они могут быть обработаны последовательно, либо, например, объединены в одну полигональную модель. На этапе подготовки сцены задается начальное положение и ориентация целевого объекта, а также вычисляются дополнительные геометрические атрибуты, необходимые для оценки качества ракурса тем или иным методом, – площадь полигонов, кривизна, площадь полигонов в текстурном пространстве и так далее.

Далее следует основная часть предлагаемого метода – этап обратного рендеринга. Сначала с помощью прямого прохода синтезируются следующие изображения:

- визуализация индексов полигонов – в каждый пиксель записывается индекс полигона, видимого в этом пикселе;
- визуализация геометрических атрибутов – в каждый пиксель записывается значение геометрического атрибута полигона, видимого в этом пикселе.

На основе обработки этих изображений вычисляется оценка качества ракурса, которая является значением целевой функции обратного рендеринга. Оптимизируемыми параметрами выступает положение и ориентация объекта (эйлеровы углы или ось и угол вращения).

Как уже было сказано, большинство существующих работ в области обратного рендеринга используют дифференцируемый рендеринг. Однако, так как в нашем случае число оптимизируемых параметров сравнительно мало, то могут быть использованы и другие методы, например, метод Нелдера-Мида. Использование безградиентных методов в рассматриваемой задаче имеет ряд преимуществ. В частности, для вычисления целевой функции можно использовать недифференцируемые операции, что расширяет выбор методов оценки ракурса. В рамках предлагаемого решения исследовалось использование как дифференцируемого рендеринга, так и безградиентных методов.

В конце этапа обратного рендеринга, значение оценки качества передается оптимизатору, завершая одну итерацию обратного рендеринга. На выходе получается матрица трансформации целевого объекта (при необходимости она может быть преобразована в матрицу трансформации виртуальной камеры), которая представляет один выбранный ракурс.

Следующим этапом, на основе визуализации индексов полигонов для полученного ракурса, производится обновление геометрических атрибутов целевого объекта. Для этого вес полигонов, видимых на выбранном ракурсе, уменьшается. За счет этого при выборе следующего ракурса процесс обратного рендеринга будет стремиться выбирать ракурсы с полигонами, которые ещё не были видны на предыдущих ракурсах.

Далее весь процесс повторяется до тех пор, пока каждый следующий ракурс уменьшает число полигонов, которые были до этого не видны, на некоторую δ . В результате на выходе формируется последовательность матриц трансформации, задающих набор выбранных ракурсов.

3.2 Оценка качества ракурса

Для оценки качества ракурсов предлагаемый метод позволяет использовать различные способы, основанные на геометрических характеристиках.

В работе [10], где также решалась задача выбора ракурса для реконструкции материалов объекта, наилучший результат продемонстрировала оценка ракурса на основе энтропии спроецированной площади полигонов, предложенная в работе [18]:

$$VE(v) = - \sum_{i=0}^N \frac{a_z(v)}{a_t(v)} \log \log \frac{a_z(v)}{a_t(v)}$$

где N - число полигонов в 3D модели, $a_z(v)$ – спроецированная площадь полигона z для ракурса v и $a_t(v)$ – спроецированная площадь всей 3D модели для ракурса v .

Под спроецированной площадью полигона a_z понимается число пикселей на изображении, принадлежащих полигону z . Для наилучшего ракурса по данной оценке все полигоны 3D модели видны с одинаковой относительной спроецированной площадью.

В рамках предлагаемого метода эта величина вычисляется за счет визуализации индексов полигонов при рендеринге. Для полученного изображения производится подсчет числа пикселей, содержащих каждое из значений индексов, что и дает спроецированную площадь. Этот способ вычисления не может быть обработан автоматическим дифференцированием. Поэтому оценка ракурса VE в предлагаемом решении может использоваться только с безградиентным обратным рендерингом.

Также в качестве оценки может выступать просто спроецированная площадь всего целевого объекта $a_t(v)$ [30]. Т.е. чем большую часть синтезированного изображения занимает целевой объект, тем лучше ракурс. В этом случае достаточно просто просуммировать пиксели изображения, в которых виден целевой объект, что позволяет использовать такую оценку как с дифференцируемым рендерингом, так и с безградиентными методами.

Схожим образом могут быть использованы и другие атрибуты геометрической модели:

- FA – суммарная площадь видимых полигонов в локальной системе координат;
- TA – суммарная площадь видимых полигонов в текстурном пространстве;
- MC – сумма модулей средней кривизны поверхности.

Наилучшим ракурсом является тот, на котором видна максимальная площадь или наиболее детальная часть модели (т.е. с наибольшей суммарной кривизной).

Расчет этих оценок также необходимо вычислять по-разному для безградиентных методов и дифференцируемого рендеринга. В случае безградиентных методов возможно получить множество всех видимых полигонов, используя индексы которых вычислить сумму требуемых атрибутов. Например, для площади в локальной системе координат:

$$FA(v) = \sum_{z \in Z_v} A_z$$

где Z – множество полигонов, видимых на ракурсе v , A_z – значение атрибута для z -го полигона.

Для дифференцируемого рендеринга операция индексации по значениям в пикселях изображения при вычислении функции потерь (целевой функции) невозможна, поэтому необходимо суммировать значения по всему изображению, содержащего в своих пикселях значения атрибутов. Это приводит к тому, что для полигона, который виден в четырех пикселях (т.е. имеет спроецированную площадь $a_z = 4$), значение атрибута, например, площади в локальных координатах, будет просуммировано 4 раза, то есть:

$$FA(v) = \sum_{z \in Z_v} a_z(v) * A_z$$

Таким образом, получается некоторое неявное «взвешивание» по спроецированной площади полигонов, – чем больше спроецированная площадь, тем больший вклад вносит данный полигон.

Для безградиентных методов нами был также исследован вариант, где используется взвешивание по относительной спроецированной площади:

$$FA(v) = \sum_{z \in Z} \frac{a_z(v)}{a_t(v)} * A_z \quad (1)$$

Чтобы иметь возможность учесть ориентацию полигонов относительно камеры мы также исследовали взвешивание по косинусу угла между вектором нормали к полигону \underline{N} и направлением на виртуальную камеру \underline{V} :

$$w_z = (0, (\underline{N} \cdot \underline{V}))$$

Для оценок с суммированием атрибутов (FA, TA, MC) это взвешивание применяется сразу при их вычислении:

$$FA(v) = \sum_{z \in Z_v} w_z(v) * A_z \quad (2)$$

Вариант с «двойным» взвешиванием – по косинусу и по спроецированной площади

$$FA(v) = \sum_{z \in Z_v} w_z(v) * a_z(v) * A_z$$

может быть также вычислен, но не имеет большого смысла, так как веса $w_z(v)$ и $a_z(v)$ зависимы.

Отметим, что предлагаемое решение не зависит от конкретного метода оценки качества ракурсов и может использовать другие существующие решения, не рассмотренные выше, если они могут быть рассчитаны с использованием только геометрической модели. Например, на основе силуэта [31] или глубины [32].

3.3 Алгоритм выбора ракурсов

Предлагаемый алгоритм выбора набора ракурсов можно разделить на две части:

- 1) внутренний цикл обратного рендеринга (оптимизации),
- 2) внешний цикл выбора набора ракурсов.

При этом внутренний цикл будет несколько отличаться в зависимости от того, используется ли дифференцируемый рендеринг (алгоритм 1) или безградиентные методы оптимизации (алгоритм 2).

В случае дифференцируемого рендеринга сразу выполняется рендеринг нужных атрибутов геометрической модели в изображение *attrib_img*, которое уже затем используются для вычисления оценки качества ракурса, в большинстве случаев простым суммированием по изображению.

При использовании безградиентной оптимизации нет ограничений на дифференцируемость операций, поэтому производится рендеринг индексов примитивов в изображение *prim_img*. Из этого изображения можно получить список всех примитивов, видимых на текущем ракурсе, и уже для них вычислять выбранную оценку ракурса.

Отметим, что в описаниях алгоритмов подразумевается параллельная обработка примитивов 3D модели. Поэтому, например, в 6-й строчке алгоритма 2 имеется в виду поэлементное умножение списка значений v некоторого атрибута примитивов на список весов примитивов W .

Алгоритмы 1 и 2 дают на выходе один ракурс. Процедура выбора нескольких ракурсов описана в алгоритме 3, который вызывает алгоритм 1 или 2 в цикле (строчка 4).

Алгоритм 1. NextView(...), Выбор ракурса с помощью дифференцируемого рендеринга

Вход:

scene – описание 3D сцены, *attrib* – имя атрибута(ов) 3D модели, используемое для расчета оценки качества ракурса, *P* – набор оптимизируемых параметров трансформации (углы эйлера или ось и угол поворота; вектор переноса), *measure* – используемая оценка качества ракурса, *cond* – условие остановки поиска одного ракурса, *lr* – отображение оптимизируемого параметра на величину его темпа обучения, δ – изменение функции потерь, меньше которого процесс оптимизации завершается, *it_{max}* – максимальное число итераций, *W* – веса примитивов 3D модели.

Выход:

$M^{4 \times 4}$ – матрица трансформации целевого объекта на подобранном ракурсе

```
1  cond ← True, it ← 0
2  while cond do
3      l ← 0,  $\hat{l}$  ← 0
4      ApplyTransform(P)
5      attrib_img ← Render(scene, attribute = attrib) // рендеринг необходимых атрибутов
6      l ← ViewpointQuality(attrib_img, measure, W) // вычисление оценки ракурса
7      OptimizationStep(l, lr) // шаг градиентного спуска
8      it ← it + 1
9      UpdateLearningRate(lr, it) // опциональное изменение темпа обучения
10     if ( $\hat{l} \neq 0$  and  $\frac{|\hat{l} - l|}{|\hat{l}|} < \delta$ ) or it ≥ itmax then
11         | cond ← False
12     else
13         |  $\hat{l}$  ← l
14     end
15 end
16 return M
```

Алгоритм 2. NextView(...), Выбор ракурса с помощью безградиентной оптимизации

Вход:

scene – описание 3D сцены, *attrib* – имя атрибута(ов) 3D модели, используемое для расчета оценки качества ракурса, *P* – набор оптимизируемых параметров трансформации (углы эйлера или ось и угол поворота; вектор переноса), *measure* – используемая оценка качества ракурса, *W* – веса примитивов 3D модели, w_a – использовать ли взвешивание по спроецированным площадям, w_n – использовать ли взвешивание по косинусу между вектором нормали и направлением на камеру

Выход:

$M^{4 \times 4}$ – матрица трансформации целевого объекта на подобранном ракурсе

```
1  cond ← True, it ← 0
2  func Objective(P):
3       $\hat{M}$  ← ApplyTransform(P)
4      prim_img ← Render(scene, attribute = prim_id) // рендеринг индексов примитивов
5      v ← ReadVisibleAttributes(attrib, prim_img, scene) // чтение атрибутов по индексам
        полигонов из изображения
6      v ← v * W
7      objective ← ViewpointQuality(v, measure, wa, wn,  $\hat{M}$ )
8      return objective * (-1)
9  end
10  $\hat{P}$  ← Minimize(Objective, P) // вызов метода оптимизации
11 M ← apply_transform( $\hat{P}$ )
12 return M
```

При выборе ракурсов из некоторого предопределенного набора заранее известно, какие полигоны геометрической модели видны на всем наборе, что позволяет ставить цель сформировать такой набор ракурсов, который будет включать каждый полигон хотя бы на одном ракурсе. В рассматриваемом же случае выбора произвольных ракурсов, такой информации нет, и легко представить случай, когда значительная часть 3D модели может быть сокрытой на любом возможном ракурсе в реальном сценарии съемки. Например, внутренняя поверхность какого-либо полого предмета. Таким образом, условие завершения работы алгоритма не может требовать видимости всех полигонов. Поэтому в качестве условия завершения формирования набора ракурсов мы используем относительное уменьшение суммы весов примитивов (алгоритм 3, строчка 10).

Алгоритм 3. Выбор набора ракурсов

Вход:

scene – описание 3D сцены, *attrib* – имя атрибута 3D модели, используемое для расчета оценки качества ракурса, *measure* – используемая оценка качества ракурса, *internal_cond* – условие остановки поиска одного ракурса, w_n – использовать ли взвешивание по косинусу между вектором нормали и направление на камеру, δ – изменение суммарного значения весов примитивов, меньше которого процесса поиска ракурсов завершается.

Выход:

\underline{M} – список матриц трансформации, описывающих положение камеры для выбранных ракурсов

```
1   $c \leftarrow 0, j \leftarrow 0, cond \leftarrow True, M \leftarrow List(),$   
    $a_t \leftarrow NumPrimitivesInMesh(scene), W \leftarrow ones(a_t), \tilde{a}_t \leftarrow a_t$   
2  while cond do  
3       $P \leftarrow InitTransform(j)$  // выбор начального положения объекта  
4       $M \leftarrow NextView(scene, attrib, measure, P, W, ...)$  // выбор одного ракурса по алгоритму 1  
       или 2  
5       $\underline{M} \leftarrow M: \underline{M}$  // добавление матрицы найденного ракурса в список  
6       $prim\_img \leftarrow Render(scene, attribute = prim\_id)$  // рендеринг индексов примитивов  
7       $\hat{a}_t \leftarrow a_t$   
8       $W \leftarrow UpdateMeshAttribute(scene, prim\_img, w_n, attrib)$  // обновление весов для  
       видимых примитивов  
9       $a_t \leftarrow sum(W)$   
10     if  $\frac{|a_t - \hat{a}_t|}{\tilde{a}_t} < \delta$  then  
11         |  $cond \leftarrow False$   
12     end  
13 end  
14 return  $\underline{M}$ 
```

Веса примитивов уменьшаются на каждом шаге после обновления (алгоритм 3, строчка 8), которое учитывает то, какие примитивы видны на последнем выбранном ракурсе. При использовании взвешивания по косинусу веса также используются при обновлении атрибутов геометрической модели после нахождения очередного ракурса:

$$mesh[attrib] = mesh[attrib] - mesh[attrib] * (0, (\underline{N} \cdot \underline{V}))$$

За счет этого видимые под углом полигоны учитываются при поиске последующих ракурсов, но с меньшим весом, т.к. они были видны на ракурсах, найденных ранее. Если взвешивание не используется, то вес видимых примитивов принимается равным нулю. В экспериментах мы использовали значение $\delta = 1\%$.

В строчке 3 алгоритма 3 производится выбор начального положения объекта. Это может быть сделано разными способами, основная задача состоит в том, чтобы обеспечить разные

начальные точки для поиска очередного ракурса, чтобы метод оптимизации мог исследовать разные части пространства решений. Удобнее выбирать положение камеры, от которого можно перейти к положению объекта. Например, может использоваться случайное положение камеры, но так, чтобы она всегда смотрела на целевой объект.

4. Экспериментальная оценка

4.1 Исходные данные

Для проведения экспериментальной оценки предложенного метода использовался набор данных Stanford-ORB [11], содержащий:

- фотографии 14 объектов в узком и широком динамическом диапазоне,
- эталонные 3D модели объектов, полученные с помощью 3D сканирования,
- HDR-панорамы освещения,
- текстуры объектов для модели материала Principled BSDF.

Текстуры были получены авторами при помощи дифференцируемого рендеринга по фотографиям, сделанным в специальных контролируемых условиях освещения, будем называть их «псевдоэталонными». Для каждого объекта заданы три текстуры – диффузный цвет, шероховатость, «металличность» (вес для смешения модели двулучевой функции рассеяния (ДФР) проводника и ДФР диэлектрика).

Из этого набора данных нами были выбраны объекты, отличающиеся геометрической формой (рис. 3), начиная от простой к более сложной.



Рис. 3. Объекты, отобранные для тестирования предложенного метода.

Fig. 3. Objects chosen for testing proposed method.

4.2 Схема проведения экспериментов

Предложенный метод и тестовая система обратного рендеринга были реализованы на базе рендер-системы Mitsuba3 [33]. Предварительная обработка полигональной модели и расчет атрибутов был реализован с помощью библиотеки libIGL [34]. Вычисление оценок качества ракурса по большей части были реализованы на GPU с использованием JIT-компилятора Dr.Jit [35], для обратного рендеринга на основе безградиентной оптимизации небольшая часть вычислений производилась на CPU.

Для каждой из 3D моделей экспериментальная оценка включала следующие этапы:

1. Предварительная обработка геометрической модели с назначением и расчетом атрибутов на каждый примитив, включающих: вес (исходно равный единице), площадь в локальной системе координат, площадь в текстурном пространстве, средняя кривизна.

2. Формирование набора ракурсов по алгоритму 3 для различных оценок ракурса для безградиентного обратного рендеринга и для дифференцируемого обратного рендеринга.
3. Прямой рендеринг 3D модели с псевдоэталонными текстурами на полученном наборе ракурсов.
4. Реконструкция текстур диффузного цвета, шероховатости и металличности для модели материала «Principled BSDF» рендер-системы Mitsuba3. В качестве входных использовались изображения, полученные на предыдущем этапе.
5. Оценка качества реконструкции по метрикам PSNR, SSIM и LPIPS [36] для трех тестовых наборов:
 - a. набор А – синтетические изображения, полученные путем рендеринга 3D модели с псевдоэталонными текстурами на ракурсах тестовой части набора данных Stanford-ORB;
 - b. набор В – реальные фотографии из тестовой части набора данных Stanford-ORB;
 - c. набор С – синтетические изображения, полученные путем рендеринга 3D модели с псевдоэталонными текстурами на 16 псевдослучайных ракурсах, полученных из трехмерной последовательности Соболя.

В качестве базового метода рассматривался выбор ракурсов из трехмерной последовательности Соболя. Для этого алгоритм 3 исполнялся без запуска процедуры обратного рендеринга (оптимизации) и использовался выбранный псевдослучайный начальный ракурс (строчка 3, алгоритм 3). В качестве измерений последовательности Соболя использовались эйлеровы углы, за счет чего такой выбор ракурсов позволил обеспечить равномерное покрытие ориентаций целевого объекта.

4.3 Результаты экспериментов

Далее приведены оценки качества реконструкции материалов на трех тестовых наборах данных – набор А (табл. 1), набор В (табл. 2), набор С (табл. 3).

Для каждого способа оценки ракурсов тестировалось три варианта – без взвешивания, со взвешиванием по косинусу угла нормали и направления на камеру (w_n) и со взвешиванием по спроецированной площади, где это имело смысл (w_a). В таблицах приведены результаты только лучшего из этих трех вариантов. Для случаев, когда использовался дифференцируемый рендеринг, это отмечено в таблице как «дифф. рендеринг», в остальных вариантах использовалась безградиентная оптимизация.

Для синтетических тестовых наборов А (табл. 1) наилучшие из методов выбора ракурсов демонстрируют преимущество на случайных выборах порядка 1.9 дБ по метрике PSNR, что согласуется с результатами, полученными авторами [13] для реконструкции нейронных полей освещенности. Для псевдослучайных синтетических ракурсов (набор данных С, табл. 3) прирост несколько меньше – около 1.2 дБ, но при этом большинство методов выбора ракурса также снижают и дисперсию PSNR по сравнению с базовым методом. При тестировании на фотографиях (набор данных В, табл. 2) использование предлагаемых методов также демонстрирует преимущество над базовым методом. По метрике SSIM все варианты показывают близкие высокие значения, по метрике LPIPS результаты в целом согласуются с таковыми по PSNR.

Среди вариантов, превосходящих базовый метод, в основном, присутствуют те, которые используют предложенные механизмы взвешивания по косинусу угла между вектором нормали к примитиву и направлением на камеру (выражение 2), которое позволяет получить больше ракурсов, где одни и те же примитивы видны под разным углом. Это, в свою очередь, может позволить получить более качественную реконструкцию материалов с бликами.

Табл. 1. Результаты на наборе синтетических изображений тестовой части Stanford-ORB (набор данных A).

Table 1. Results for synthetic images of testing part of Stanford-ORB (dataset A).

Метод оценки ракурса	Среднее число ракурсов	Среднее PSNR, дБ ↑	Среднее SSIM ↑	Среднее LPIPS ↓
Базовый	8.6	33.99 ± 1.98	0.967 ± 0.012	0.019 ± 0.012
Средняя кривизна, w_n	11.6	35.86 ± 2.23	0.971 ± 0.013	0.014 ± 0.010
Спроецированная площадь, w_n	11	35.72 ± 2.61	0.971 ± 0.015	0.015 ± 0.011
Площадь в текстурном пространстве, w_n	11.6	35.84 ± 2.50	0.971 ± 0.015	0.015 ± 0.012
Энтропия спроецированной площади	8.4	34.23 ± 2.19	0.967 ± 0.015	0.019 ± 0.013
Площадь полигонов, w_n	11	35.86 ± 1.97	0.972 ± 0.012	0.014 ± 0.010
Площадь полигонов, w_n , дифф. рендеринг	11	35.66 ± 2.49	0.971 ± 0.013	0.015 ± 0.010
Площадь в текстурном пространстве, дифф. рендеринг	9	35.27 ± 2.25	0.969 ± 0.015	0.016 ± 0.012
Спроецированная площадь, дифф. рендеринг	11.6	35.51 ± 2.02	0.970 ± 0.013	0.016 ± 0.011

Табл. 2. Результаты на наборе реальных изображений (фотографий) тестовой части Stanford-ORB (набор данных B).

Table 2 Results for real images (photos) of testing part of Stanford-ORB (dataset B).

Метод оценки ракурса	Среднее PSNR, дБ ↑	Среднее SSIM ↑	Среднее LPIPS ↓
Базовый	30.36 ± 1.51	0.962 ± 0.013	0.025 ± 0.014
Средняя кривизна, w_n	31.17 ± 2.17	0.966 ± 0.015	0.021 ± 0.014
Спроецированная площадь, w_n	30.95 ± 2.27	0.965 ± 0.017	0.022 ± 0.015
Площадь в текстурном пространстве, w_n	31.05 ± 2.11	0.965 ± 0.016	0.022 ± 0.015
Энтропия спроецированной площади	30.39 ± 1.64	0.962 ± 0.015	0.025 ± 0.016
Площадь полигонов, w_n	31.07 ± 1.98	0.966 ± 0.014	0.022 ± 0.014
Площадь полигонов, w_n , дифф. рендеринг	31.07 ± 2.06	0.966 ± 0.014	0.022 ± 0.014
Площадь в текстурном пространстве, дифф. рендеринг	30.85 ± 1.95	0.964 ± 0.016	0.023 ± 0.016
Спроецированная площадь, дифф. рендеринг	30.97 ± 1.94	0.965 ± 0.014	0.023 ± 0.014

Табл. 3. Результаты на наборе синтетических изображений с псевдослучайными ракурсами (набор данных C).

Table 3. Results for images with pseudorandom viewpoints (dataset C).

Метод оценки ракурса	Среднее PSNR, дБ ↑	Среднее SSIM ↑	Среднее LPIPS ↓
Базовый	32.81 ± 3.05	0.959 ± 0.015	0.019 ± 0.010
Средняя кривизна, w_n	33.95 ± 2.62	0.962 ± 0.014	0.015 ± 0.007
Спроецированная площадь, w_n	33.84 ± 2.70	0.962 ± 0.015	0.016 ± 0.007
Площадь в текстурном пространстве, w_n	34.07 ± 2.52	0.962 ± 0.014	0.016 ± 0.008
Энтропия спроецированной площади	32.69 ± 3.10	0.959 ± 0.016	0.019 ± 0.010
Площадь полигонов, w_n	34.05 ± 2.44	0.963 ± 0.013	0.015 ± 0.007
Площадь полигонов, w_n , дифф. рендеринг	34.00 ± 2.84	0.963 ± 0.014	0.015 ± 0.007
Площадь в текстурном пространстве, дифф. рендеринг	33.45 ± 2.74	0.961 ± 0.014	0.016 ± 0.008
Спроецированная площадь, дифф. рендеринг	33.78 ± 2.40	0.962 ± 0.013	0.016 ± 0.008

Число выбранных ракурсов в обучающем наборе при использовании обратного рендеринга в среднем имеет значения от 8.4 до 11.6. На рис. 4 показан пример набора ракурсов, выбранных по оценке, основанной на суммарной средней кривизне.

С увеличением числа ракурсов растет и качество реконструкции, что ожидаемо. Но при этом метод на основе энтропии спроецированной площади, выбирающий в среднем наименьшее число ракурсов, демонстрирует небольшое увеличение качества реконструкции по сравнению с базовым методом, выбирающим немногим большее количество ракурсов. Это может служить дополнительным подтверждением того, что использование обратного рендеринга позволяет выбрать ракурсы, более подходящие для реконструкции материалов.



Рис. 4. Пример набора ракурсов, выбранных с помощью оценки качества ракурса по суммарной средней кривизне.

Fig. 4. Example set of viewpoints selected using the total mean curvature view quality measure.

В сравнении с реконструкцией по синтетическим изображениям из всего обучающего набора Stanford ORB, включающего 60 ракурсов, наилучший из вариантов с выбором ракурсов проигрывает около 1.5 дБ по метрике PSNR на синтетических тестовых наборах А и С и показывает почти такой же результат на фотографиях (набор В). При этом среднее число используемых для реконструкции изображений для предложенных вариантов в среднем меньше в ~6 раз, что позволяет значительно снизить время реконструкции. Для GPU Nvidia RTX 2070 Super реконструкция на 60 ракурсах в нашей реализации занимает порядка 2-х часов. А при использовании наборов ракурсов, полученных предложенным методом (в среднем 10 ракурсов), для реконструкции требуется в 8 раз меньше времени – в среднем 15 минут.

Время получения набора ракурсов с помощью предложенного метода при использовании обратного рендера на основе безградиентной оптимизации (методом Нелдера-Мида в реализации библиотеки `scipy`) занимает в среднем 1.5 минуты. Для дифференцируемого рендеринга на подбор ракурсов требуется в среднем 15 минут. Таким образом, дополнительные вычисления для выбора ракурсов при использовании безградиентной оптимизации составляют лишь 10% от времени реконструкции. При использовании дифференцируемого рендеринга время выбора ракурсов примерно равно времени реконструкции по выбранным ракурсам. Но, если сравнить суммарное время выбора ракурсов и реконструкции с временем реконструкции на всех 60 ракурсах из обучающей части набора данных Stanford-ORB, то разница достигает порядка 4-8 раз в зависимости от используемого механизма обратного рендеринга и меры оценки качества ракурса.

5. Заключение

Предложенный метод выбора ракурсов для задачи реконструкции материалов на основе обратного рендеринга с использованием 3D модели целевого объекта в качестве входных данных показал свою применимость.

Предложенный метод позволяет использовать произвольные способы оценки качества ракурсов, опирающиеся на геометрическую модель целевого объекта. Это обеспечивается за счет возможности применения разных подходов к обратному рендерингу, включая дифференцируемый рендеринг и безградиентную оптимизацию.

Для протестированных способов оценки качества ракурсов использование предложенного метода позволяет получить небольшой набор ракурсов (порядка 8-11 изображений), позволяющих произвести реконструкцию с достаточно высоким качеством, при использовании чисто синтетических обучающих данных превосходящим 34 дБ по метрике PSNR на синтетических тестовых наборах данных и около 31 дБ на реальных тестовых данных. При этом небольшое количество ракурсов приводит к снижению вычислительных затрат на проведение реконструкции.

Список литературы / References

- [1]. Mildenhall B, Srinivasan PP, Tancik M, Barron JT, Ramamoorthi R, Ng R. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*. 2021 Dec 17;65(1):99-106. DOI: 10.1145/3503250.
- [2]. Kerbl B, Kopanas G, Leimkühler T, Drettakis G. 3D Gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*. 2023 Aug 1;42(4):139-1. DOI: 10.1145/3592433.
- [3]. Reiser C, Peng S, Liao Y, Geiger A. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. *Proceedings of the IEEE/CVF international conference on computer vision 2021* (pp. 14335-14345). DOI: 10.48550/arXiv.2103.13744.
- [4]. Seo S, Han D, Chang Y, Kwak N. Mixerf: Modeling a ray with mixture density for novel view synthesis from sparse inputs. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2023* (pp. 20659-20668). DOI: 10.1109/CVPR52729.2023.01979.

- [5]. Li Z, Wang D, Chen K, Lv Z, Nguyen-Phuoc T, Lee M, Huang JB, Xiao L, Zhu Y, Marshall CS, Ren Y. LIRM: Large Inverse Rendering Model for Progressive Reconstruction of Shape, Materials and View-dependent Radiance Fields. Proceedings of the Computer Vision and Pattern Recognition Conference 2025 (pp. 505-517). DOI: 10.48550/arXiv.2504.20026.
- [6]. Ummerhofer B, Agrawal S, Sepulveda R, Lao Y, Zhang K, Cheng T, Richter S, Wang S, Ros G. Objects with lighting: A real-world dataset for evaluating reconstruction and rendering for object relighting. 2024 International Conference on 3D Vision (3DV) 2024 Mar 18 (pp. 137-147). IEEE. DOI: 10.1109/3DV62453.2024.00097.
- [7]. Zhao S, Jakob W, Li TM. Physics-based differentiable rendering: from theory to implementation. ACM siggraph 2020 courses 2020 Aug 17 (pp. 1-30). DOI:10.1145/3388769.3407454.
- [8]. Luan F, Zhao S, Bala K, Dong Z. Unified shape and svbrdf recovery using differentiable monte carlo rendering. Computer Graphics Forum 2021 Jul (Vol. 40, No. 4, pp. 101-113). DOI: 10.1111/cgf.14344.
- [9]. Vicini D, Speierer S, Jakob W. Path replay backpropagation: Differentiating light paths using constant memory and linear time. ACM Transactions on Graphics (TOG). 2021 Jul 19;40(4):1-4. DOI: 10.1145/3450626.3459804.
- [10]. Sanzharov V., Frolov V. Viewpoint selection for texture reconstruction with inverse rendering. Proceedings of GraphiCon 2023 (pp. 66-78). DOI: 10.20948/graphicon-2023-66-77.
- [11]. Kuang Z, Zhang Y, Yu HX, Agarwala S, Wu E, Wu J. Stanford-orb: a real-world 3d object inverse rendering benchmark. Advances in Neural Information Processing Systems. 2023 Dec 15;36:46938-57. DOI: 10.48550/arXiv.2310.16044.
- [12]. Voynov O, Bobrovskikh G, Karpyshev P, Galochkin S, Ardelean AT, Bozhenko A, Karmanova E, Kopanov P, Labutin-Rymsho Y, Rakhimov R, Safin A. Multi-sensor large-scale dataset for multi-view 3D reconstruction. InProceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2023 (pp. 21392-21403). DOI:10.1109/CVPR52729.2023.02049.
- [13]. Xiao W, Cruz RS, Ahmedt-Aristizabal D, Salvado O, Fookes C, Lebrat L. Nerf director: Revisiting view selection in neural volume rendering. InProceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2024 (pp. 20742-20751). DOI: 10.1109/CVPR52733.2024.01960.
- [14]. Bonaventura X, Feixas M, Sbert M, Chuang L, Wallraven C. A survey of viewpoint selection methods for polygonal models. Entropy. 2018 May 16;20(5):370. DOI: 10.3390/e20050370.
- [15]. Zhang Y, Fei G. Overview of 3D scene viewpoints evaluation method. Virtual Reality & Intelligent Hardware. 2019 Aug 1;1(4):341-85. 10.1016/j.vrih.2019.01.001.
- [16]. Zeng S, Geng G, Zhou M. Automatic Representative View Selection of a 3D Cultural Relic Using Depth Variation Entropy and Depth Distribution Entropy. Entropy. 2021 Nov 23;23(12):1561 DOI: 10.3390/e23121561.
- [17]. Neumann L, Sbert M, Gooch B, Purgathofer W. Viewpoint quality: Measures and applications. Proceedings of the 1st eurographics workshop on computational aesthetics in graphics, visualization and imaging. aire-la-vile: The eurographics association press 2005 (pp. 185-192).
- [18]. Vázquez PP, Feixas M, Sbert M, Heidrich W. Automatic view selection using viewpoint entropy and its application to image-based modelling. Computer Graphics Forum 2003 Dec (Vol. 22, No. 4, pp. 689-700). Oxford, UK and Boston, USA: Blackwell Publishing, Inc. DOI:10.1111/j.1467-8659.2003.00717.x.
- [19]. Feixas M, Sbert M, González F. A unified information-theoretic framework for viewpoint selection and mesh saliency. ACM Transactions on Applied Perception (TAP). 2009 Feb 25;6(1):1-23. DOI:10.1145/1462055.1462056.
- [20]. Zhou W, Jia J. Training deep convolutional neural networks to acquire the best view of a 3D shape. Multimedia Tools and Applications. 2020 Jan;79(1):581-601. DOI: 10.1007/s11042-019-08107-w.
- [21]. Song R, Zhang W, Zhao Y, Liu Y. Unsupervised multi-view CNN for salient view selection and 3D interest point detection. International Journal of Computer Vision. 2022 May;130(5):1210-27. 10.1007/s11263-022-01592-x.
- [22]. Pan X, Lai Z, Song S, Huang G. Activenerf: Learning where to see with uncertainty estimation. European Conference on Computer Vision 2022 Oct 23 (pp. 230-246). Cham: Springer Nature Switzerland. DOI: 10.48550/arXiv.2209.08546.
- [23]. Li Y, Li R, Li Z, Guo R, Tang S. OptiViewNeRF: Optimizing 3D reconstruction via batch view selection and scene uncertainty in Neural Radiance Fields. International Journal of Applied Earth Observation and Geoinformation. 2025 Feb 1;136:104306. DOI: 10.1016/j.jag.2024.104306.
- [24]. Mendoza M, Vasquez-Gomez JI, Taud H, Sucar LE, Reta C. Supervised learning of the next-best-view for 3d object reconstruction. Pattern Recognition Letters. 2020 May 1;133:224-31. DOI: 10.1016/j.patrec.2020.02.024.

- [25]. Wang T, Xi W, Cheng Y, Han H, Yang Y. RL-NBV: A deep reinforcement learning based next-best-view method for unknown object reconstruction. *Pattern Recognition Letters*. 2024 Aug 1;184:1-6. DOI: 10.1016/j.patrec.2024.05.014.
- [26]. Zhang Z, Xu F, Zhang M. Peering into the Unknown: Active View Selection with Neural Uncertainty Maps for 3D Reconstruction. *arXiv preprint arXiv:2506.14856*. 2025 Jun 17. DOI: 10.48550/arXiv.2506.14856.
- [27]. Li Y, Kuang Z, Li T, Hao Q, Yan Z, Zhou G, Zhang S. Activesplat: High-fidelity scene reconstruction through active gaussian splatting. *IEEE Robotics and Automation Letters*. 2025 Jun 16. DOI: 10.48550/arXiv.2410.21955.
- [28]. Jin L, Zhong X, Pan Y, Behley J, Stachniss C, Popović M. Activegvs: Active scene reconstruction using gaussian splatting. *IEEE Robotics and Automation Letters*. 2025 Mar 26. DOI: 10.48550/arXiv.2412.17769.
- [29]. Chen L, Zhan H, Chen K, Xu X, Yan Q, Cai C, Xu Y. Activegamer: Active gaussian mapping through efficient rendering. *Proceedings of the Computer Vision and Pattern Recognition Conference 2025* (pp. 16486-16497). DOI: 10.48550/arXiv.2501.06897.
- [30]. Plemenos D., Benayada M. Intelligent display techniques in scene modelling. new techniques to automatically compute good views. *Proceedings of the International Conference GraphiCon 1996*.
- [31]. Polonsky O, Patané G, Biasotti S, Gotsman C, Spagnuolo M. What's in an image? Towards the computation of the "best" view of an object. *The Visual Computer*. 2005 Sep;21(8):840-7. DOI: 10.1007/s00371-005-0326-y.
- [32]. Secord A, Lu J, Finkelstein A, Singh M, Nealen A. Perceptual models of viewpoint preference. *ACM Transactions on Graphics (TOG)*. 2011 Oct 22;30(5):1-2. DOI: 10.1145/2019627.201962.
- [33]. W. Jakob, S. Speierer, N. Roussel, M. Nimier-David, D. Vicini, T. Zeltner, B. Nicolet, M. Crespo, V. Leroy, Z. Zhang. *Mitsuba 3 renderer*, 2022. URL: <https://mitsuba-renderer.org>.
- [34]. A. Jacobson, D. Panozzo, C. Schüller, O. Diamanti, Q. Zhou, S. Koch, J. Dumas, A. Vaxman, N. Pietroni, S. Brugger, K. Takayama, W. Jakob, N. De Giorgis, L. Rocca, L. Sacht, K. Walliman, O. Sorkine-Hornung, T. Schneider et al. 2023. *libigl - A simple C++ geometry processing library* URL: <https://libigl.github.io>.
- [35]. Jakob W, Speierer S, Roussel N, Vicini D. Dr. jit: A just-in-time compiler for differentiable rendering. *ACM Transactions on Graphics (TOG)*. 2022 Jul 22;41(4):1-9. DOI: 10.1145/3528223.3530099.
- [36]. Zhang R, Isola P, Efros AA, Shechtman E, Wang O. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition 2018* (pp. 586-595). DOI: 10.1109/CVPR.2018.00068).

Информация об авторах / Information about authors

Вадим Владимирович САНЖАРОВ – кандидат физико-математических наук, младший научный сотрудник лаборатории компьютерной графики и мультимедиа факультета Вычислительной математики и кибернетики ВМК МГУ. Сфера научных интересов: прямой и обратный рендеринг, моделирование оптических систем.

Vadim Vladimirovich SANZHAROV – Cand. Sci. (Phys.-Math.), junior researcher at Graphics and media laboratory, Faculty of computational mathematics and cybernetics at Lomonosov Moscow State University. Research interests: forward and inverse rendering, optical systems modeling.

Владимир Александрович ФРОЛОВ – кандидат физико-математических наук, научный сотрудник лаборатории компьютерной графики и мультимедиа факультета Вычислительной математики и кибернетики ВМК МГУ, старший научный сотрудник отдела №2 ИПМ им. М.В. Келдыша РАН. Сфера научных интересов: реалистичная компьютерная графика, моделирование освещённости, разработка программных систем оптического моделирования, параллельные и распределённые вычисления.

Vladimir Aleksandrovich FROLOV – Cand. Sci. (Phys.-Math.), researcher at Graphics and media laboratory, Faculty of computational mathematics and cybernetics at Lomonosov Moscow State University, senior researcher at department №2 at Keldysh Institute of Applied Mathematics RAS.

Research interests: realistic computer graphics, light transport simulation, optical simulation software systems, parallel and distributed computations.

Владимир Александрович ГАЛАКТИОНОВ – доктор физико-математических наук, профессор, главный научный сотрудник ИПМ им. М.В. Келдыша РАН. Сфера научных интересов: компьютерная графика, оптическое моделирование.

Vladimir Aleksandrovich GALAKTIONOV – Dr. Sci. (Phys.-Math.), professor, principal researcher at Keldysh Institute of Applied Mathematics RAS. Research interests: computer graphics, optical simulation.

DOI: 10.15514/ISPRAS-2025-37(4)-21



A Distributed Framework for Large-Scale Data Analysis Using Bio-Inspired Sensory-Motor Algorithms

*D.P. Potapov, ORCID: 0009-0007-6232-3138 <sablist99@bk.ru>
S.M. Staroletov, ORCID: 0000-0001-5183-9736 <serg_soft@mail.ru>
Polzunov Altai State Technical University,
46, prospect Lenina, Barnaul, Altai region, 656038, Russia.*

Abstract. In the process of detecting anomalies or deviations from expected behavior in continuously streaming data is complex and necessitates the development of effective models that can adaptively retrain over time. The human brain serves as a prime example of such a system, as it continuously learns throughout life, with past experiences that once seemed erroneous gradually becoming integrated into commonplace knowledge. While modern neural network models have made significant advancements in recognizing text and images, they have diverged considerably from the original neuron models and no longer represent a singular algorithm akin to that which our brains utilize. Networks such as LSTM (Long Short-Term Memory) can account for both distant and immediate past information; however, they exhibit limitations in their retrainability. We align with the theories proposed by Jeff Hawkins, a prominent researcher in the field of bio-inspired intelligence, whose team is developing innovative cortical algorithms that emulate current research on the functioning of the intelligent brain. In this context, vision and hearing can be conceptualized as sensors, with the data they provide being integrated within the model to generate continuous predictions for each input signal. In our article, we explore contemporary theories on this subject and present a custom implementation of these concepts using the Erlang programming language.

Keywords: brain-inspired model; cortical algorithms; anomalies in streaming data.

For citation: Potapov D.P., Staroletov S.M. A distributed framework for large-scale data analysis using bio-inspired sensory-motor algorithms. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025., pp. 103-116. DOI: 10.15514/ISPRAS-2025-37(4)-21.

Распределенная платформа для анализа масштабных данных с использованием биоподобных сенсомоторных алгоритмов

Д.П. Потапов, ORCID: 0009-0007-6232-3138 <sablist99@bk.ru>

С.М. Старолетов, ORCID: 0000-0001-5183-9736 <serg_soft@mail.ru>

АлмГТУ им. И.И. Ползунова

Россия, 656038, Алтайский край, г. Барнаул, пр. Ленина, 46.

Аннотация. Процесс обнаружения аномалий или отклонений от ожидаемого поведения в непрерывно поступающих данных является сложным и требует разработки эффективных моделей, которые могут адаптивно переобучаться со временем. Человеческий мозг служит ярким примером такой системы, так как он постоянно учится на протяжении всей жизни, а прошлый опыт, который когда-то казался ошибочным, постепенно интегрируется в обыденные знания. Хотя современные модели нейронных сетей достигли значительных успехов в распознавании текста и изображений, они значительно отклонились от оригинальных моделей нейронов и больше не представляют собой единственный алгоритм, аналогичный тому, который использует наш мозг. Специальные искусственные сети, такие как Long Short-Term Memory (LSTM), могут учитывать как длительную, так и непосредственную информацию; однако они имеют ограничения в своей способности к переобучению. В данном исследовании мы согласуемся с теориями, предложенными Джеффом Хокинсом, выдающимся исследователем в области биоподобного интеллекта, чья команда разрабатывает инновационные кортикальные алгоритмы, которые используют современные исследования функционирования интеллектуального мозга. В этом контексте зрение и слух можно концептуализировать как сенсоры, данные от которых интегрируются в модель для генерации непрерывных прогнозов для каждого входного сигнала. В нашей статье мы исследуем современные теории по этой теме и представляем собственную реализацию этих концепций с использованием языка программирования Erlang.

Ключевые слова: биоподобная модель; кортикальные алгоритмы; поиск аномалий в потоковых данных.

Для цитирования: Потапов Д.П., Старолетов С.М. Распределенная платформа для анализа масштабных данных с использованием биоподобных сенсомоторных алгоритмов. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 103-116 (на английском языке). DOI: 10.15514/ISPRAS-2025-37(4)-21.

1. Introduction

The neocortex is the uppermost layer of the mammalian brain, which is primarily responsible for higher-order cognitive functions, including intelligence. The study of the columnar organization of the neocortex, as proposed by Jeff Hawkins' theory [1-2], represents a significant avenue for understanding the functioning of the human brain. Hawkins posited a model based on the premise that the neocortex comprises thousands of columns of vertical structures that operate in parallel and coordinate their activities to process information. Research into the columnar organization of the neocortex within the framework of this theory enhances our understanding of various aspects of brain function, such as perception, memory, problem-solving, and the formation of concepts.

Furthermore, research on Hawkins' theories is of great relevance not only for advancing fundamental scientific knowledge about the brain but also from a practical perspective for the development of innovative data analysis technologies. In the context of anomaly detection, data analysis plays a crucial role. Anomalous data can be characterized as unusual or unexpected, and their detection is a critical task across numerous fields, including finance, cybersecurity, and medicine. Given that neocortical columns are involved in pattern detection and prediction, they can be instrumental in identifying abnormal or unusual patterns in data [3].

The objective of our work is to gain a deeper understanding of Hawkins' theories and the proposed cortical algorithms, as well as to present our own implementation of these concepts. Our primary interest lies in developing large, high-load distributed systems; therefore, we selected the Erlang programming language for our implementation.

The structure of the following article is organized as follows: In Section 2, we conduct a comprehensive review of the topic of cortical algorithms and the columnar organization of neurons. In Section 3, we propose an architectural framework for a system that implements such algorithms. In Section 4, we examine our understanding of the mathematical principles proposed by Hawkins' team. In Section 5, we address implementation challenges. In Section 6, we discuss the learning visualization module. Finally, we conclude in Section 7 and provide a link to the repository containing the code.

2. Related Works

The emergence of intelligence from the cells (neurons) in the brain remains a profound mystery. As more pieces of this intricate puzzle are uncovered each year, it appears that we are moving further away from a comprehensive understanding of the brain, rather than closer to it [4].

When examining a cross-section of the neocortex, one can observe a complex and detailed structure. However, the architectural details appear remarkably similar across different regions of the cortex. For instance, the area of the cortex responsible for vision resembles the area dedicated to touch, which in turn resembles the area associated with language. Mountcastle suggests that the reason these areas exhibit such similarities is that they all perform analogous functions [5]. What differentiates them is not their internal operations, but rather their connections. If a specific area of the cortex is connected to the eyes, it facilitates vision; if the same area is connected to the ears, it enables hearing; and if these areas are linked to other regions, they contribute to higher cognitive processes, such as language. Mountcastle posits that if one can ascertain the primary function of any given part of the neocortex, it may lead to a broader understanding of the entire system.

There are, however, regions that are more challenging to delineate, particularly those involved in high-level thinking and planning. These areas are interconnected by bundles of nerve fibers. Some of the connections between regions are hierarchical, suggesting that information flows in an orderly manner from one area to another, akin to a flowchart (as discussed in Hawkins' first book [1]). Conversely, there are other connections that appear to lack a clear order, indicating that information may flow to various parts simultaneously.

In each area examined, scientists have identified cells that project to ancient parts of the brain involved in movement. For example, the visual areas, which receive input from the eyes, send signals to the primitive brain regions that control eye movement. Similarly, the auditory areas, which receive input from the ears, project to the brain regions responsible for head movement. Consequently, moving one's head alters auditory perception, just as moving one's eyes affects visual perception. The evidence suggests that the complex circuitry found throughout the neocortex is engaged in sensorimotor tasks.

In his book, "A Thousand Brains: A New Theory of Intelligence" [2], Jeff Hawkins elucidates why neurons in the cerebral cortex have organized themselves into communities known as "cortical columns." This concept is grounded in the later works of Mountcastle [6] (Fig. 1). There are approximately 150,000 of these columns in the neocortex. Hawkins and his team discovered that each column constituting the cerebral cortex creates its own model of the world. Consequently, rather than a singular model, there exist thousands of models, and our perception emerges as a collective decision made by the cortical columns through a voting mechanism. They assert that it is not artificial neural networks, but rather the principles governing the neocortex that they have uncovered, which will serve as the foundation for the future development of artificial intelligence.

In a subsequent article [7], Hawkins explores the rationale behind neurons possessing thousands of synapses and how this relates to sequence memory within the neocortex. According to his model, each neuron can process and store a small segment of a sequence, with the entire sequence being encoded through synaptic connections at the network level of neurons. The article elucidates the significance of the thousands of synapses connecting to a single neuron in the processing of sequential information. In particular, Hawkins explains that the multitude of synapses enables a

neuron to detect and remember temporal patterns of activation, which is crucial for the contextual understanding of sequences. The article also addresses the role of synaptic connections and the plasticity of neural networks in the formation and retention of memory. Hawkins emphasizes that each synaptic connection can possess varying weights and can change based on neuronal activity. This adaptability allows the brain to flexibly adjust and enhance the efficiency of information processing.

718 V. B. Mountcastle

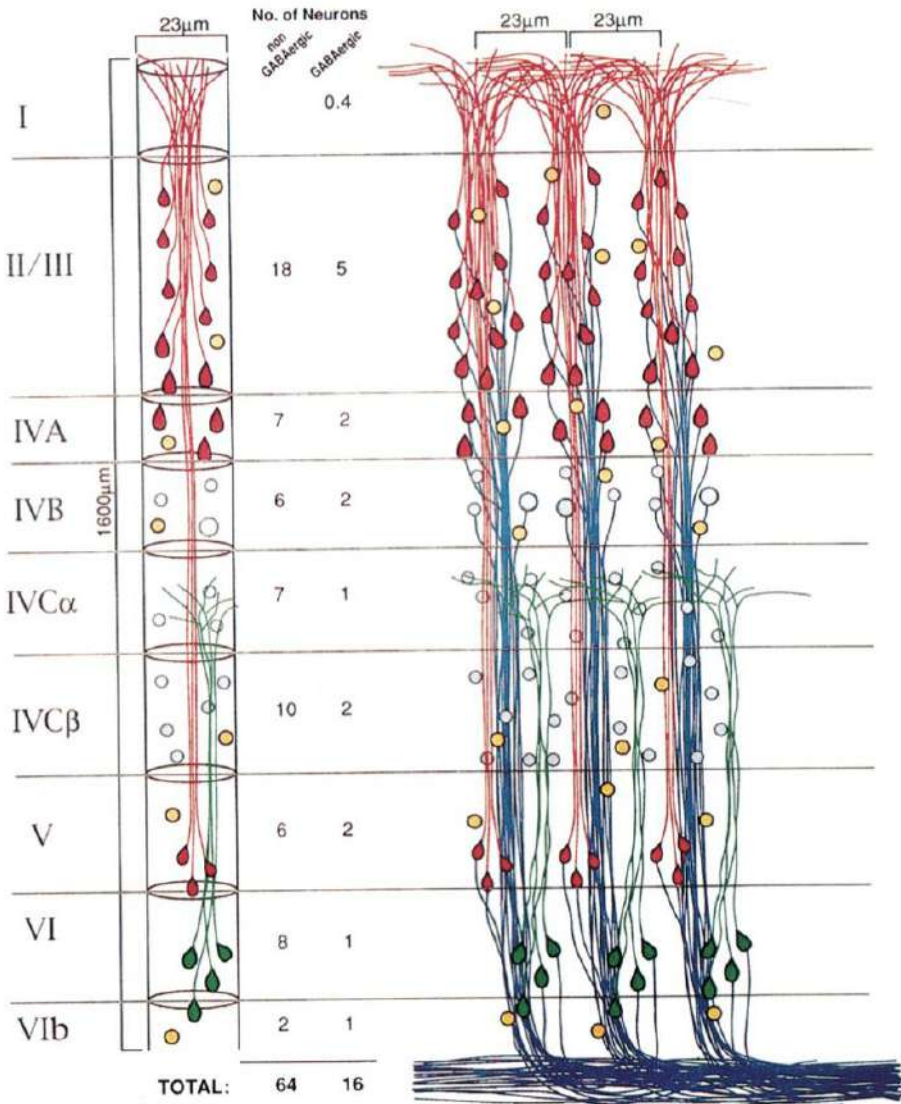


Fig. 1. Distributed organization of columns, model proposed in [5-6].

In the following article [8], Hawkins and his team provide a more in-depth exploration of the neocortex model. The principle that neocortical regions are organized into columns and layers is upheld. The connections between layers are predominantly perpendicular to the surface, suggesting a columnar functional organization. Additionally, some layers exhibit long-range excitatory lateral

106

connections, indicating interactions between columns. Similar interaction patterns exist across regions, but their precise roles remain unclear. Emphasis is placed on a multilayer structure that facilitates robust learning and object recognition. Each column integrates changing inputs over time to develop comprehensive predictive models of observed objects. Excitatory lateral connections between columns enable the network to infer objects more rapidly based on partial knowledge from neighboring columns. Because columns integrate inputs across both time and space, the network learns models of complex objects that extend well beyond the receptive fields of individual cells.

The discussed network model introduces a novel function for cortical columns. It is proposed that a representation of the location relative to the perceived object is computed in specialized layers of each column. This location signal is then fed as an input to the network, where it is combined with sensory data. The sample model consists of two layers and one or more columns. Simulations demonstrate that, using Hebbian-like learning rules, small single-column networks can learn to recognize hundreds of objects.

When two neurons fire simultaneously, the connection between them is strengthened. As we learn new information, these connections are reinforced, while forgetting leads to a weakening of the connections. This fundamental concept was proposed by Donald Hebb in the 1940s and is now referred to as Hebbian learning [9].

Multi-column networks are capable of recognizing objects with significantly fewer movements of sensory receptors. Given the ubiquity of columnar and laminar connectivity patterns throughout the neocortex, there is reason to believe that columns and regions possess more powerful recognition and modeling capabilities than previously understood.

The following distinctions can be made between the approach under consideration and traditional artificial neural networks. Modern artificial neural networks are primarily based on the "strength" of the synapse. In contrast, the "Thousand Brains" theory posits that the mere presence of a synapse is sufficient, and its "strength" does not significantly influence the accuracy of predictions. Additionally, while modern neural networks employ biological terminology, they do not aim to accurately replicate the processes occurring in the neocortex. For instance, they often overlook the roles of dendrites and dendritic spikes.

As illustrated in Fig. 2, the bio-neuron model closely resembles the cells of the Long Short-Term Memory (LSTM) model [10], which is a sort of a recurrent artificial neural network characterized by gates that incorporate signal forgetting coefficients for both long-term and short-term perspectives. Such networks are also applicable for anomaly detection [11], although they necessitate relatively powerful computational resources. Generally, these networks exhibit limitations in retrainability, although there are research implementations of retrainable networks that maintain references to memory areas where input data are stored [12], which is analogous to the location used in sensory-motor algorithms.

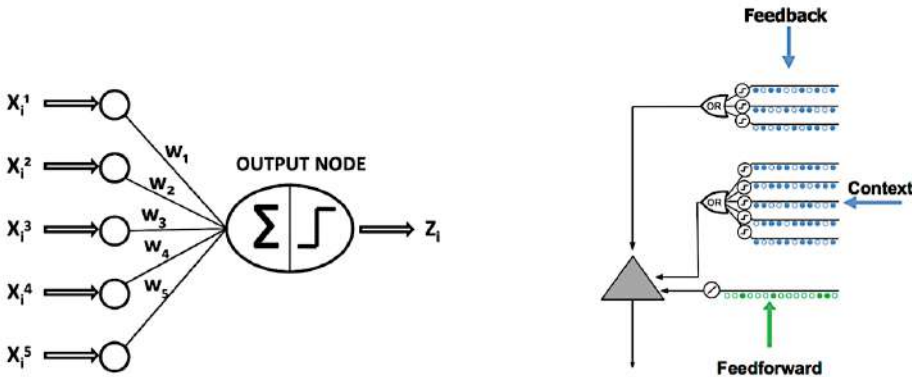


Fig. 2. The difference between a perceptron and a pyramidal bio- neuron according to Hawkins.

The current work is not the first in a series of studies on this topic by the authors of this article. In the very first work, we present our own understanding of Hawkins’s original theory (when he had only written the whitepaper), and propose our own algorithms for learning and visualizing cortical columns [13]. In paper [14], we developed a simple model for predicting characters in a string using cortical algorithms. In paper [15], we presented a preliminary outline of this article. In article [16], we conducted a review of cortical algorithms and proposed ideas for integrating multiple sensors.

3. Architectural Foundation

3.1 Model of a biological neuron and one cortical layer

The basic unit of information storage in the cortical column is the synapse, which indicates the presence or absence of connections (see Fig. 3 for reference).

From a biological point of view, the structure of the neocortex column can be represented in a diagram shown in Fig. 4.

From a biological perspective, the structure of the neocortical column can be represented in the diagram shown in Fig. 4. The lighter representation of the mini-column reflects its status as a conventional unit. The organization of cells within the column may not correspond precisely to their physical arrangement; however, it is essential to understand that these cells belong to a single conventional group. For the sake of clarity in modeling, the cells are organized into "mini-columns." The synapse operates under a similar rationale: it is defined as a connection between a dendrite and an axon, serving as a conventional unit that lacks a direct physical manifestation (at least within the framework of this model).

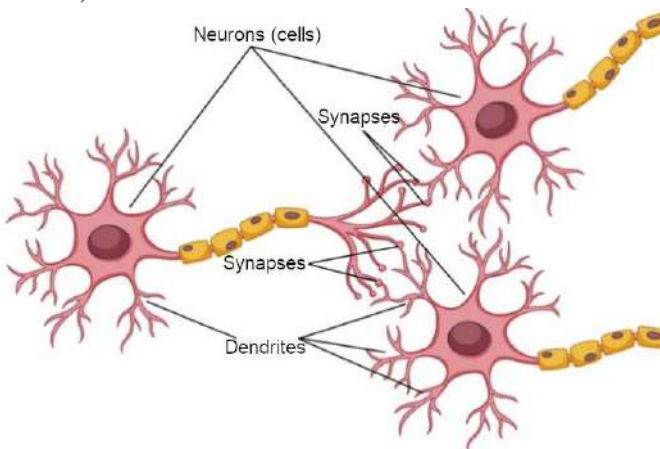


Fig .3. Model of a biological neuron.

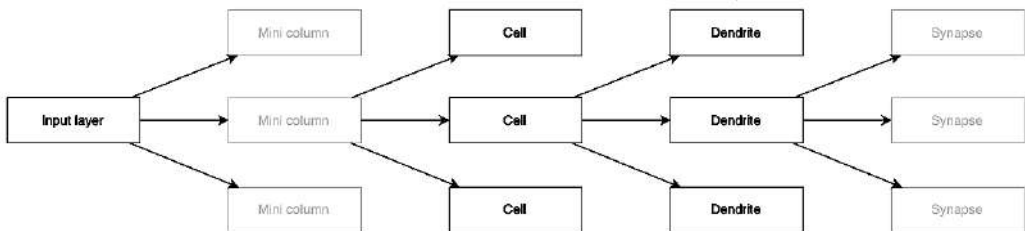


Fig. 4. Model of one cortical layer.

Synapses are essentially represented by a pair of parameters: (1) a constancy value and (2) a synapse weight. The constancy value is a continuous variable ranging from 0 to 1, indicating the strength of the connection and whether it exists at all. Additionally, there is a threshold value that determines the existence of a synapse. If the constancy value exceeds the threshold, the synapse weight is set to one; otherwise, it is set to zero. During the learning process, the constancy values of synapses change, which affects the "stability" of the synapses. If a particular pattern is repeated consistently, the constancy value will increase, indicating that the pattern is being recognized correctly. It is advisable to initialize the synapse values using values derived from a Poisson distribution. Experimental findings suggest that this distribution should be centered at the point 0.12, ensuring that only some values exceed the specified threshold and form a synapse.

3.2 Multilevel model of the cortical column

The general model of the system is shown in Fig. 5, where the following color Indication is used:

- Green: direct signals that travel along the proximal dendrites. These signals serve to directly activate neurons.
- Blue: lateral signals that propagate along the basal dendrites. These signals function to depolarize cells, effectively predicting activation. Cells with lateral connections are activated more rapidly and can suppress the activation of cells that lack such connections.
- Red: feedback signals that travel along the apical dendrites. This feedback originates from the outer layer. If a cell in the outer layer has been activated, it can predict the activation of cells in the input layer by depolarizing them through the apical dendrite.

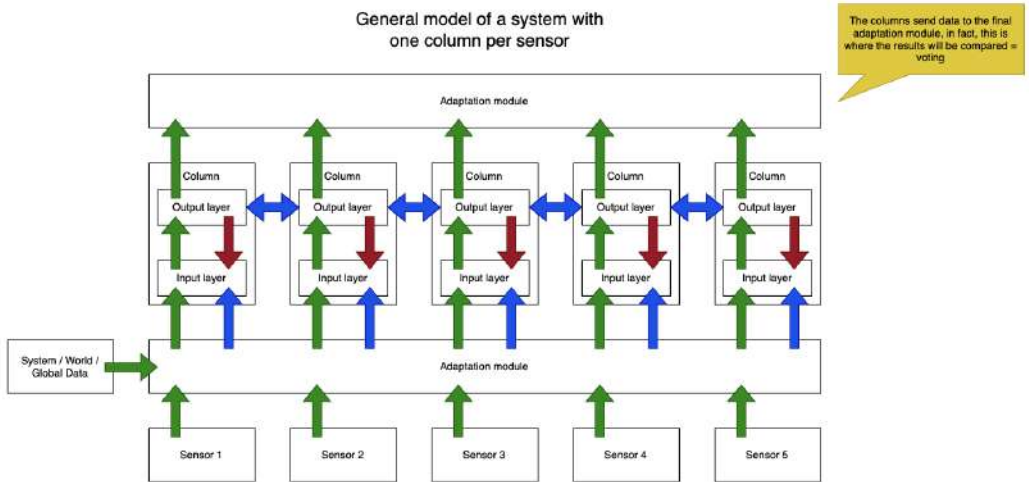


Fig. 5. General model of a two-level distributed system.

It is important to note that columns can communicate not only with neighboring columns within a single area of the neocortex but also with columns in other areas (see Fig. 6).

Moreover, there may even be a situation where the connections between columns form a fully connected graph.

Typically, there are several cortical columns per "sensor", which in turn "communicate" with each other and "make" a collective decision on recognizing the activation pattern. Taking this aspect into account, the diagram can be transformed as shown in Fig. 7.

Interaction of columns

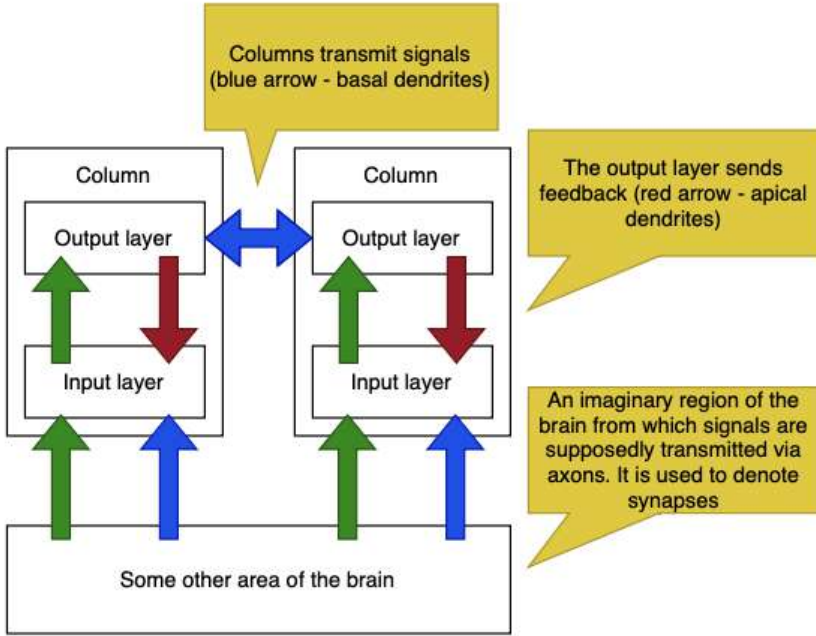


Fig. 6. Interaction of columns with other areas of the neocortex.

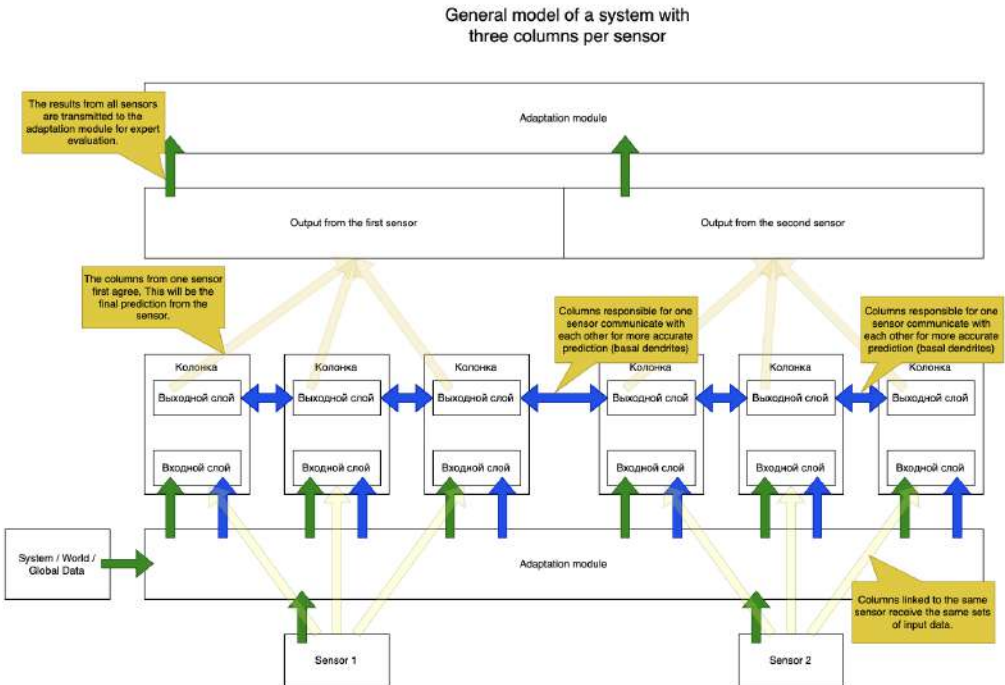


Fig. 7. Multiple columns per sensor.

4. Mathematical Foundation

4.1 Model of neuron state transitions

A neuron can be in three states: inactive, depolarized, active. It is advisable to store the states in separate data structures. For depolarized cells, a binary matrix π^{in} can be used, the matrix is filled in according to formula (1):

$$\pi_{ij}^{in} = \begin{cases} 1, & \text{if } \exists d \left(L \cdot D^{ijd,in} \geq \theta_b^{in} \right) > 0 \\ \text{else} & 0 \end{cases} \quad (1)$$

where i, j are the conditional coordinates of the neuron; d is the dendrite; L is the location signal; $D^{ijd,in}$ is the binary matrix of existing synapses on the d -th dendrite of the ij -th neuron; θ_b^{in} is the dendrite activation threshold.

That is, those cells will be predicted that have at least one dendrite that has a sufficient number of synapses with the input signal.

The transition of a neuron to the active stage occurs according to formula (2):

$$a_{ij}^{in} = \begin{cases} 1, & \text{if } j \in W^{in} \text{ and } \pi_{ij}^{in} > 0 \\ 1, & \text{if } j \in W^{in} \text{ and } \sum_i \pi_{ij}^{in} = 0 \\ \text{else} & 0 \end{cases} \quad (2)$$

where a_{ij}^{in} is a binary matrix with active cells; j is the mini-column number; W^{in} is the set of columns to which a direct signal has been received.

In other words, the previously predicted cells that have received a direct signal will become active. Additionally, the mini-columns that have received a direct signal will also become active, even if no individual cell within the mini-column has been predicted.

Active cells in the input layer send a direct signal to the output layer to activate it. The activation of the output layer cells will occur according to the following formulas (3)-(6):

$$o_k^{out,t} = \sum_{i,j} I[f_{ijk} \geq \theta_c^{out}] a_{ij}^{in,t} \quad (3)$$

where k is the index of the output layer cell; $o_k^{out,t}$ is the number of cells that send a direct signal to the k -th cell of the output layer; I is the indicator function; f_{ijk} is the constancy value of the synapse between the ij -th cell of the input layer and the k -th cell of the output layer; θ_c^{out} is the activation threshold of the synapse between layers; $a_{ij}^{in,t}$ are the active cells at step t .

$$W^{out,t} = \{k | o_k^{out,t} \geq \theta_p^{out}\} \quad (4)$$

where $o_k^{out,t}$ are the output layer cells with a direct signal at step t ; θ_p^{out} is the threshold number of cells that must send a signal to the output layer at the k -th cell for the signal to become strong enough.

$$a_i^{out,t} = \begin{cases} 1, & \text{if } i \in W^{out,t} \text{ and } p_i^{out,t-1} \geq \xi_{t-1}^{out} \\ \text{else} & 0 \end{cases} \quad (5)$$

where $a_i^{out,t}$ are the active cells of the output layer at step t ; $p_i^{out,t-1}$ is the number of active basal dendrites at the previous step; ξ_{t-1}^{out} is the threshold number of basal dendrites required for activation.

$$p_i^{out,t-1} = \sum_d I[\tilde{A}^{out} \cdot \tilde{D}^{id,out} \geq \theta_b^{out}] \quad (6)$$

where $\tilde{A}^{out} \cdot \tilde{D}^{id,out}$ are the existing synapses between the output layer and the dendrite; θ_b^{out} is the dendrite activation threshold.

In other words, an output cell will become active if it has enough direct and basal connections. If the number of cells with lateral support is less than some s , then $\xi_{t-1}^{out} = 0$.

4.2 Training

For training, a specific output layer pattern is selected that will be reinforced when new sensory and location signals are received. New synapses will be formed to facilitate the activation of this pattern, while those that produce false positives will be eliminated.

Thus, the training of dendritic segments will occur according to the following formula (7):

$$\Delta D^{ijd,in} = p_{in}^+ D^{ijd,in} \circ L^t - p_{in}^- D^{ijd,in} \circ (1 - L^t) \quad (7)$$

where $\Delta D^{ijd,in}$ is the coefficient by which the synapse constancy value will change; p_{in}^+ is the coefficient for rewarding the “correct” synapse; $D^{ijd,in}$ is a binary matrix with dendrite synapses; L^t is the location signal at time t ; p_{in}^- is the coefficient for decreasing the “incorrect” synapse.

4.3 Data storage capacity

The capacity of a cortical column network is influenced by four key parameters:

1. Representational space of the network.
2. Number of minicolumns.
3. Number of neurons in the output layer.
4. Number of cortical columns.

Considering that the diameter of cortical columns ranges from 300 to 600 μm , and the diameter of a minicolumn ranges from 30 to 60 μm , we can estimate that a cortical column contains between 150 and 250 minicolumns. Thus, a network comprising 150 minicolumns, with 16 cells per minicolumn and 10 simultaneously active cells, can potentially represent approximately 10^{15} sensory representations and 10^{16} location representations.

If an output neuron connects to an excessive number of input neurons, it may be erroneously activated by a pattern that it was not specifically trained on. Consequently, the capacity of the network is constrained by the size of the output layer.

For instance, a column with 4,096 cells in the output layer is capable of remembering up to 400 objects, as illustrated in the graph provided in article [8].

4.4 Towards anomaly detection

The requirements for an anomaly detection system are as follows:

- Predictions must be made online; that is, the algorithm must identify a state $x(t)$ as normal or abnormal before receiving the subsequent state $x(t+1)$.
- The algorithm must learn continuously without the need to store the entire data stream.
- The algorithm must operate in an unsupervised and unattended manner, meaning it should function without labeling the data or manually tuning the parameters.
- The algorithms must be capable of adapting to a dynamic environment and concept drift, as the underlying statistics of the data stream are often non-stationary.
- The algorithms must detect anomalies as early as possible.
- The algorithms must minimize both false positives and false negatives.

The discussed model provides a framework for sequential learning in the cortex. Implementations operate in real time and are well-suited for prediction tasks. The model continuously learns and

models the spatio-temporal characteristics of its input data, but it does not directly model anomalies or produce a ready-to-use anomaly score.

At each time step, the input data $x(t)$ is fed into the model, triggering the activation of cells. The resulting vector $a(x(t))$ is a sparse binary vector representing the current input signal. The discussed model serves as the sequence memory component, which models the temporal patterns in $a(x(t))$ and outputs a prediction in the form of another sparse vector $\pi(x(t))$. Thus, $\pi(x(t))$ represents the prediction for $a(x(t+1))$.

Subsequently, the prediction errors $S(t)$ are calculated. Using the probabilistic model $S(t)$, the probability that the system is in an abnormal state $L(t)$ is determined. The threshold of this probability dictates whether an anomaly will be detected (here, we need to reconsider the information from work [17] and apply it to our implementation).

5. Implementation

The core of the system is implemented in the functional language Erlang [18], which allows working with pattern matching and lightweight processes.

5.1 Data representation structures

To store layer data, we utilize a dictionary (key-value pairs), where each layer corresponds to one dictionary.

For the input layer, the key represents the rank of the input binary data, while the value is a mini-column, which is also a dictionary. In the mini-column dictionary, the key is the cell GUID, and the value is another dictionary representing the dendrites. Dendrites, in turn, are structured as a dictionary where the key corresponds to the bit number of the lateral signal, and the value represents the synapse.

A synapse is defined by its GUID, along with the values for constancy and weight. The output layer is similarly structured as a dictionary, where the key is the bit number, and the value is a pair consisting of the cell GUID and the dictionary of dendritic branches. The structure of the dendrites in the output layer mirrors that of the input layer.

The dendritic branches of the output layer are sized according to the output layer itself, as the lateral support in this layer refers back to itself.

Separate dictionaries, labeled FeedForward (green arrows in Fig. 5) and FeedBack (red arrows in Fig. 5), store the synapses between the cells of the input and output layers. For convenience, the rank number of the external dictionary is also stored. The overall storage structure is as follows:

$$\text{map} \lll \langle \langle \text{range}, \text{id}(in) \rangle, \langle \text{range}, \text{id}(out) \rangle \rangle, \text{permanenceValue} \ggg$$

To maintain the state of cell predictions at the current iteration, we employ a data scheme that is structurally similar. However, this scheme specifically stores data related to the predicted cells, including the mini-columns containing the predicted cells, the predicted cells themselves, and the dendrites that led to their activation. Additionally, for each cell, we store the status of the active apical dendrite (the GUID of the dendrite itself, if applicable).

To represent the state of cell activity at the current iteration, we use a structure that consists of a dictionary organized by mini-columns, where the value contains a list of GUIDs of active cells. In this case, there is no need to store dendrites again, as they are already included in the dictionary of predicted cells. It should be noted that if a cell is active but not present in the dictionary of predicted cells, it was activated due to the absence of predicted cells in the mini-column, resulting in the activation of the entire column [13].

In this context, all data is considered global within the process and is stored in the global Process Dictionary, allowing any function to access the data at any time without the need to pass parameters to functions, thereby preventing an increase in the call stack.

5.2 Transition between states

The entry point to the activation process is the receipt of a location signal. This signal consists of a set of numbers, and its size corresponds to the size of the dendrite of the input layer neuron. All cells in the input layer are examined to identify dendrites that contribute to cell depolarization. The presence of an apical dendrite is also considered in this determination, as it lowers the activation threshold of the dendrite.

The next stage involves receiving a direct signal, which is a set of numbers corresponding to the identifiers of the mini-columns. Cells that were previously depolarized are activated. Furthermore, if there are depolarized cells with an active apical dendrite within the mini-column, only these cells are activated. If no predicted cells are present in the mini-column and a direct signal is received, then all cells within the mini-column will become active.

The activation of input layer cells leads to the activation of output layer cells. In this process, lateral support from neighboring cortical columns and the activation pattern from the previous time step are taken into account. However, if weak support prevents the activation of a sufficient number of cells in the output layer, then lateral support is disregarded.

5.3 Training

The core operates in three modes:

- Active learning
- Passive learning
- No learning

Active learning refers to the scenario in which we know in advance the object of interest and establish the corresponding activation pattern in the output layer. Consequently, during the learning process, the connections that facilitate the activation of this pattern will be strengthened.

When training the input layer, dendrites that contribute to the depolarization of cells that subsequently become active are rewarded. Conversely, if a dendrite leads to depolarization but the cell does not become active, that dendrite is "punished."

The renewal of synapses between layers and the lateral support synapses occurs in a similar manner.

6. Visualization Module

The visualization module interface includes functionality that allows the researcher to work with signals, including editing and sending both direct signals and location signals. Additionally, there is a visual representation of data, which enables the analysis of processes occurring during the operation of the system.

Specifically, the module facilitates:

- Analysis of the state of the cells in the input layer.
- Analysis of the state of the dendrites of the cells in the input layer:
- Presence of synapses.
- Presence of active dendrites.
- Analysis of connections between layers (FeedForward and FeedBack).
- Analysis of the activity of the output layer.

The first array of dots in the left column of Fig. 8 represents the input layer, where cells are organized into mini-columns that correspond to the visual columns. Inactive cells are depicted in gray, depolarized cells in blue, and active cells in orange. The second array of dots represents FeedBack connections. The third array of dots represents FeedForward connections. The fourth array of dots

corresponds to the output layer, which is a one-dimensional array of cells displayed in multiple lines for clarity. Inactive cells are shown in gray, while predicted cells are represented in orange.

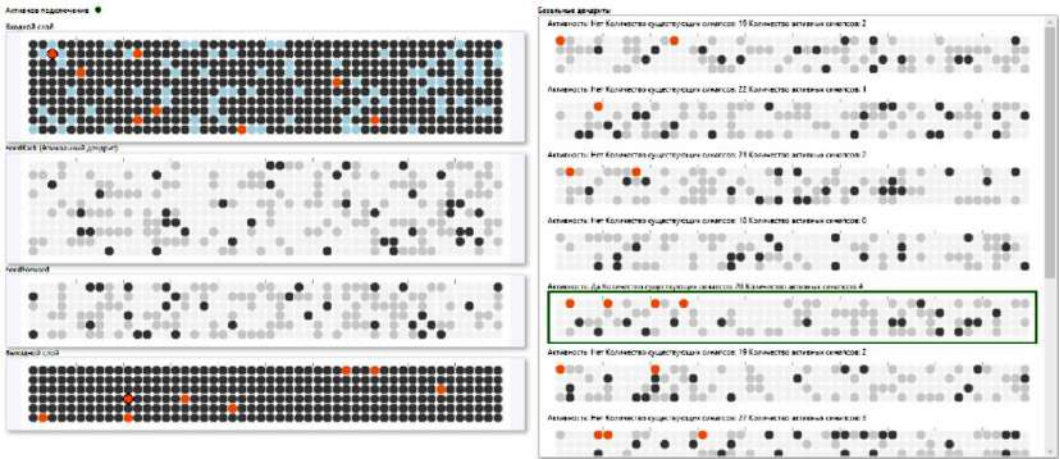


Fig. 8. Visualization of training our model.

7. Conclusion

The work involved a comprehensive review of the literature and scientific articles authored by Jeff Hawkins and the Numenta research group. In accordance with the proposed theory, we developed software consisting of two modules: the core of the system and a visualization module.

Overall, the developed system represents a promising tool for analyzing big data to detect anomalies. It can be applied across various fields, such as healthcare, finance, and manufacturing, to enhance decision-making processes and increase efficiency.

In the future, this research can be expanded in the following directions:

- Implementation of a data adaptation module, which will include systems for converting data from various sensors into a sparse binary vector and decomposing it into two components: location data and sensory data.
- Investigation of new sensory-motor algorithms and the neocortex representation of location data.
- Exploration of the functions of different types of neurons in the neocortex, such as grid cells.
- Development of new methods for visualizing and interpreting data to improve the understanding of analysis results and facilitate anomaly detection.
- Integration of the system with other data analysis systems to create comprehensive solutions for big data analysis.

The current version of the discussed software is available on GitHub [19].

Finally, it should also be noted that recently Russian translations of Hawkins' books have appeared, available through marketplaces [20]. However, to really understand the theory, we need to read the articles of the authors' research group and try to write our own implementations.

References

- [1]. Hawkins J., Blakeslee S. On intelligence. Macmillan, 2004.
- [2]. Hawkins J. A Thousand Brains: A New Theory of Intelligence. Basic books, 2021.

- [3]. Lavin A., Ahmad S. Evaluating real-time anomaly detection algorithms-the Numenta anomaly benchmark. In 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA). IEEE, pp. 38-44, 2015.
- [4]. Hubel D. H. The brain. *Scientific American*, Vol. 241, no 3, pp. 45-53, 1979.
- [5]. Edelman G. M., Mountcastle V. B. The mindful brain: Cortical organization and the group-selective theory of higher brain function. MIT press, 1982.
- [6]. Mountcastle V. B. The columnar organization of the neocortex. *Brain: a journal of neurology* 120, no. 4, pp. 701-722, 1997.
- [7]. Hawkins J., Ahmad S. Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in neural circuits*, vol 10, p. 23, 2016.
- [8]. Hawkins J., Ahmad S., Cui Y. A theory of how columns in the neocortex enable learning the structure of the world. *Frontiers in neural circuits* Vol. 11 p. 95079, 2017.
- [9]. Hebb D.O. *The Organization of Behavior*. New York: Wiley & Sons, 1949.
- [10]. Hochreiter S, Schmidhuber J. Long short-term memory. *Neural computation* Vol 9, no. 8 pp. 1735-1780, 1997.
- [11]. Lindemann B., Maschler B., Sahlab N., Weyrich M. A survey on anomaly detection for technical systems using LSTM networks. *Computers in Industry* Vol. 131 p. 103498, 2021.
- [12]. Wang H., Li M., Yue X. IncLSTM: incremental ensemble LSTM model towards time series data. *Computers & Electrical Engineering* Vol. 92, pp. 107156, 2021.
- [13]. Staroletov S. A hierarchical temporal memory model in the sense of Hawkins. In 2021 IEEE Ural-Siberian Conference on Computational Technologies in Cognitive Science, Genomics and Biomedicine (CSGB). IEEE, pp. 470-475, 2021.
- [14]. Potapov D. P., Tselebrovsky O. B., Staroletov S. M. Designing a bio-inspired memory-prediction model according to Hawkins (in Russian). In *Modern Digital Technologies Barnaul*, June 01, p. 243. 2023. Available: <https://www.elibrary.ru/item.asp?id=54479906>, accessed Jul. 15, 2025.
- [15]. Potapov D. P., Staroletov S. M. Design of a multilevel distributed system for analysis by bio-inspired sensory-motor algorithms (in Russian). In *Modern Digital Technologies Barnaul*, June 03, 2024. Available: <https://www.elibrary.ru/item.asp?id=68573379>, accessed Jul. 15, 2025.
- [16]. Staroletov S. M. Review of the current state of cortical algorithms and their application to real-time signal analysis. (In Russian). *System Administrator* Vol. 11, no. 240 p. 82-87, 2022. Available: <https://www.elibrary.ru/item.asp?id=50021376>.
- [17]. Ahmad S., Lavin A. Purdy S., Agha Z. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* Vol. 262, pp. 134-147, 2017.
- [18]. Armstrong J. *Programming Erlang: Software for a Concurrent World*, 2013.
- [19]. Software implementation. [Online]. Available: <https://github.com/sablist99/ThousandBrains>, accessed Jul. 15, 2025.
- [20]. Hawkins J. Volume 112. *Super brain. Revolution in understanding human and artificial intelligence*. Sber library, Alpina PRO, 2024. ISBN 978-5-206-00224-9. Available: <https://sberuniversity.ru/research/library/46924>, accessed Jul. 15, 2025.

Информация об авторах / Information about authors

Данил Петрович ПОТАПОВ – ассистент кафедры прикладной математики. Сфера научных интересов: кортикальные алгоритмы, аддитивные технологии.

Danil Petrovich POTAPOV – assistant lecturer at the department of Applied Mathematics. Research interests: cortical algorithms, additive manufacturing.

Сергей Михайлович СТАРОЛЕТОВ – кандидат физико-математических наук, доцент. Сфера научных интересов: формальная верификация, model checking, киберфизические системы, операционные системы.

Sergey Mikhailovich STAROLETOV – Cand. Sci. (Phys.-Math.), associate professor. Research interests: formal verification, model checking, cyber-physical systems, operating systems.

DOI: 10.15514/ISPRAS-2025-37(4)-22



Автоматическая генерация рецензий к коду: эволюция инструкций и интеллектуальная фильтрация

В.В. Качанов, ORCID: 0000-0002-9371-6483 <vkachanov@ispras.ru>

*Институт системного программирования им. В.П. Иванникова РАН,
Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.*

Аннотация. В работе рассматривается задача автоматической генерации рецензий к исходному коду. Предложен и апробирован подход на основе больших языковых моделей, значительно повышающий практическую ценность рецензий кода за счет фокуса на применимости, настройки инструкций и интеллектуальной фильтрации. Рассмотрено систематическое инкрементальное применение следующих стратегий: пошаговое рассуждение, структурированный вывод, расширение контекста, обучение с примерами. Использован подход со специальной инструкцией к языковой модели для интеллектуального ранжирования и фильтрации нерелевантных комментариев. Использование предложенного подхода к конструированию инструкций к языковой модели позволило улучшить точность применимых рецензий в 2.5 раза по сравнению с базовой до 37%.

Ключевые слова: автоматизация рецензирования исходного кода; большие языковые модели; качество набора данных.

Для цитирования: Качанов В.В. Автоматическая генерация рецензий к коду: эволюция инструкций и интеллектуальная фильтрация. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 117–132. DOI: 10.15514/ISPRAS-2025-37(4)-22.

Automatic Code Review Generation: Instruction Evolution and Intelligent Filtering

V.V. Kachanov, ORCID: 0000-0002-9371-6483 <vkachanov@ispras.ru>

*Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*

Abstract. Code review is essential for software quality but labor-intensive in distributed teams. Current automated comment generation systems often rely on evaluation metrics focused on textual similarity. These metrics fail to capture the core goals of code review, such as identifying bugs, security flaws, and improving code reliability. Semantically equivalent comments can receive low scores if worded differently, and inaccurate suggestions can create confusion for developers. This work aims to develop an automated code review generator focused on producing highly relevant and applicable feedback for code changes. The approach leverages Large Language Models, moving beyond basic generation. The core methodology involves the systematic design and incremental application of sophisticated prompt engineering strategies. Key strategies include step-by-step reasoning instructions, providing the model with relevant examples (few-shot learning), enforcing structured output formats, and expanding contextual understanding. Crucially, a dedicated intelligent filtering stage is introduced: a LLM-as-a-Judge technique acts as an evaluator to rigorously rank generated comments and filter out irrelevant, redundant, or misleading suggestions before presenting results. The approach was implemented and tested using the Qwen/Qwen2.5-Coder-32B-Instruct model. Evaluation by original code authors demonstrated significant improvements. The optimal prompt strategy yielded a 2.5 times increase in the proportion of applicable reviews (reaching 37%) and a 1.6 times increase in good comments (reaching 61%) compared to a baseline. Providing examples enhanced comment quality, and the evaluator filter proved highly effective in boosting output precision. These results represent a substantial advance towards generating genuinely useful, actionable feedback. The approach significantly enhances the practical utility and user experience of automated code review tools for software developers by prioritizing relevance and applicability.

Keywords: code review automation; large language models; dataset quality.

For citation: Kachanov V.V. Automatic Code Review Generation: Instruction Evolution and Intelligent Filtering. *Trudy ISP RAN/Proc. ISP RAS*, vol. 37, issue 4, part 2, 2025, pp. 117-132 (in Russian). DOI: 10.15514/ISPRAS-2025-37(4)-22.

1. Введение

В условиях современных распределённых команд разработки программного обеспечения, проведение рецензирования исходного кода (code review) продолжает оставаться крайне важным аспектом обеспечения качества, несмотря на свою внушительную трудоемкость. Традиционный подход, требующий тщательного ручного анализа каждого набора изменений или патча, неизбежно сопряжён со значительными временными затратами и когнитивной нагрузкой на разработчиков. В свете этих вызовов, научное и инженерное сообщество проявляет растущий интерес к системам автоматической генерации комментариев к коду. Эти системы рассматриваются как перспективный инструмент для оптимизации процесса рецензирования, повышения его эффективности и масштабируемости.

В начальной фазе исследований данной проблематики, доминирующую роль занимали подходы, основанные на методах информационного поиска, которые фокусировались преимущественно на извлечении подходящих комментариев из обширных предварительно собранных репозиториях существующих рецензий [1]. Однако, достижения в области глубокого обучения (deep learning) инициировали значительную трансформацию парадигмы, сместив акцент исследований в сторону генеративных моделей искусственного интеллекта. Работа коллектива исследователей под руководством Tufano [2], с архитектурой трансформера T5 [3] в основе, получила дальнейшее развитие и была усовершенствована благодаря внедрению стратегий предварительного обучения моделей на масштабных гибридных корпусах, сочетающих фрагменты программного кода с пояснениями на

естественном языке [4]. Следующее поколение моделей, таких как CodeReviewer [5] и AUGER [6] интегрировали в свою структуру специализированные механизмы предварительного обучения (pretrain), целенаправленно настроенные именно на задачу рецензирования кода для повышения точности и релевантности. Альтернативным подходом оставался метод информационного поиска, продолживший свое развитие в работе CommentFinder [7], предлагающей оптимизированные решения на базе поисковых алгоритмов. В совокупности, всё многообразие этих исследовательских усилий наглядно иллюстрирует сложный характер современных подходов к решению актуальной проблемы автоматизации в инженерной практике создания программных продуктов.

Тем не менее, несмотря на очевидный прогресс в области генерации рецензий к исходному коду, существующие методологии оценки качества автоматически создаваемых комментариев зачастую вызывают серьёзные вопросы. Практика показывает, что они в значительной степени опираются на поверхностные текстовые метрики (такие, как BLEU и ROUGE), которые, по сути, игнорируют истинные, фундаментальные цели процесса рецензирования кода – а именно, выявление потенциальных дефектов, уязвимостей безопасности, архитектурных просчётов и, в конечном итоге, реальное улучшение качества и надёжности программного кода [5, 8]. Более того, установлено, что недостаточно точные комментарии не просто бесполезны, но и потенциально вредны, так как способны вносить путаницу в процесс разработки и дезориентировать программистов [8-9]. Ключевым недостатком стандартных метрик является и то, что они систематически "штрафуют" и занижают оценку технически корректных и семантически эквивалентных комментариев, если их лексическая форма отличается от предоставленного эталонного варианта. Именно поэтому, центральной целью настоящего исследования является разработка генератора комментариев для анализа патчей и коммитов, основанного на использовании больших языковых моделей (LLM) и сфокусированного в первую очередь на обеспечении высокой степени релевантности и практической применимости даваемых рекомендаций. Этот акцент обусловлен убеждением, что метрики применимости (включая точность обнаружения реальных ошибок, оценку практической полезности предлагаемых исправлений, уровень ложных срабатываний) гораздо адекватнее отражают подлинную цель рецензирования – улучшение качества исходного кода. В рамках данной работы также освещаются: архитектура предложенной системы, поэтапная эволюция и настройка используемых инструкций/промптов (prompt engineering), механизмы фильтрации выходных данных для повышения их качества и результаты комплексной оценки эффективности генератора на репрезентативных наборах реальных производственных данных.

2. Существующие решения

Был проведен анализ существующих подходов к генерации рецензий к исходному коду. Авторы работы [10] описывают процесс генерации комментариев при помощи LLM, а также применение подхода LLM-as-a-Judge для автоматической оценки качества рецензий и сравнения нескольких. Авторы использовали подходы, называемые 'few-shot' и 'reasoning' [11]. Также в открытый доступ предоставлен исходный код и набор данных, на котором проводилась оценка качества. В публикации было введено 10 параметров качества рецензий по которым происходит как оценка, так и написание: Readability, Relevance, Explanation Clarity, Problem Identification, Actionability, Completeness, Specificity, Contextual Adequacy, Brevity. Основной единицей, с которой работает разработанный подход, являются методы и функции.

В работе [12] описывается промышленная система для автоматизированного code review в компании ByteDance – BitsAI-CR. Выделяются 4 базовые категории, на которые можно разбить основные полезные комментарии в рецензиях. В качестве контекста для модели используют измененный метод в унифицированном формате. Для оценки эффективности в первую очередь используют метрику точности предложенных комментариев, так как для

практического применения системы важно избежать предоставления пользователю большого количества низкокачественных комментариев, даже если это происходит в ущерб полноте. В качестве примера открытой модели в исследовании используется большая языковая модель Qwen/Qwen2.5-Coder-32B-Instruct [13].

В работе [8] подчеркивается критическая важность качества обучающих данных для эффективности моделей генерации комментариев к коду, предлагаются методы их очистки. Авторы фокусируются на устранении «шумных» (малоинформативных) комментариев из открытых наборов данных, используя LLM для фильтрации и сохранения только качественных рецензий. Показано, что модели, обученные на очищенных данных, демонстрируют улучшение (на 12–13% по метрике BLEU) в сходстве с эталонными человеческими комментариями. Авторы работы [9] предлагают подход *DesiView*. Он использует большие языковые модели, чтобы выделять рецензии, указывающие на реальные дефекты, классифицируя комментарии и обучая модель только на полезных данных. Такой подход позволяет генерировать более точные и целенаправленные комментарии, сфокусированные на существенных проблемах кода. Обе работы эмпирически доказывают, что предварительная обработка и строгий отбор тренировочных данных являются ключевыми факторами для повышения качества вывода генеративных систем в области автоматизированного рецензирования кода.

3. Предлагаемые методы

3.1 Базовая инструкция

В работе [10] предложена инструкция к модели, генерирующей рецензии к исходному коду, названная *LLM-Reviewer*, которой в роли контекста исходного кода подается унифицированный формат измененного метода. В инструкции применяется метод, заключающийся в добавлении к инструкции небольшого числа примеров работы из «обучающего» набора, однако эти примеры подбираются случайным образом (*few-shot learning*). К недостаткам подхода можно отнести отсутствие структурированного вывода [14] работы модели, что осложняет понимание и выделение конкретных замечаний. Так, модель не указывает к какой конкретно строке относится предложение по исправлению, что не удобно пользователю, особенно, когда комментариев несколько, а генерируемая рецензия – достаточно длинная. Также при отсутствии стандартизированного вывода модель не всегда предлагает исправленный вариант кода.

К минусам можно также отнести отсутствие контекста, хотя бы уровня класса или файла, что приводит к генерации замечаний такого вида "Ensure that all necessary modules (``numpy``, ``matplotlib.pyplot``, ``sklearn.metrics``) are imported at the beginning of the file. This is crucial for the code to run without errors." или "The function ``set_seed(seed)`` is called but not defined within the snippet. Ensure that ``set_seed(seed)`` is defined elsewhere in the codebase or import it from an appropriate module.". Такие комментарии явно указывают, что ограничение контекста модели только одним методом повышает количество ошибочных рецензий.

3.2 Структурированный вывод

Структурированный вывод (*structured output*) – это технология, позволяющая большим языковым моделям генерировать выходные данные в строго заданном формате, таком как JSON, XML или таблицы, вместо произвольного текста. Такой метод обеспечивает предсказуемость, машинную читаемость и упрощает интеграцию с внешними системами (например, с базами данных или с прикладными системами) [14-15].

Указание структуры в запросе с конкретным примером далеко не всегда гарантирует, что итоговый вывод модели будет таковым или в целом корректным форматом данных, например, словарем в языке Python. Применение структурированного вывода значительно

повышает надежность вывода модели в формате данных JSON или других сложных структур в ответе модели.

В библиотеке LangChain [16] представлены методы интеграции структурированного вывода, реализованные на базе библиотеки Pydantic [17]. Проверка вывода исключает пропущенные поля и неверные типы данных. Ограничение формата снижает риск генерации некорректной информации.

Использование описанного подхода может привести и к нежелательным эффектам, например, к снижению качества генерации, так как строгие форматы могут ухудшать качество генерации, увеличивать время работы и обработки ответа модели. К тому же не все модели поддерживают и хорошо работают с режимом структурированного вывода.

Для решаемой задачи определен и добавлен следующий структурированный вывод для модели – каждая рецензия должна состоять из комментария, предложенного исправленного кода, уровня важности (от 1 до 5) и номера строки, к которой относится комментарий.

3.3 Расширение контекста

Для улучшения восприятия моделью исходного кода и внесенных изменений было предложено расширить контекст с уровня конкретного метода до уровня файла. Так, в случае добавления в коммите нового файла следует подавать модели его целиком и рецензии будут писаться ко всему файлу, а не к отдельным методам. Если файл был изменен, то для полностью новых методов подавать в модель этот метод и все изменения в файле в формате unidiff. Для измененных методов предлагается подавать на вход модели все в соответствующем файле. Чтобы дать модели возможность точно указать место, к которому относится замечание, к целевым методам добавляется нумерация строк. Для изменений нумеруются все не удаленные строки, например:

```
111: def sum( x: int, y: int):  
-     print(f'{x} + {y} = {x + y}')
```

```
112: +     logging.info(f'{x} + {y} = {x + y}')
```

```
113:     return x + y
```

Внедрение указанных усовершенствований привело к значительному сокращению абсолютно не релевантных комментариев. Тем не менее остаточные проблемы сохраняют свою актуальность, включая как критичные случаи, так и избыточные рекомендации, не оптимальные в данном контексте. К числу наиболее распространённых групп относятся:

1. Предложения по улучшению, уже реализованные в анализируемом коде, например, добавить работу с контекстным менеджером `with open(...)`, при явном его наличии;
2. Предложения добавить избыточные проверки, например, проверки наличия ключей в словарях там, где это гарантировано контекстом;
3. Предложения вставить каждый вызов метода в операторы перехвата исключений (`try-except`), игнорируя принцип разумной достаточности;
4. Предложения, противоречащие проектным практикам (например, замена метода `'print'` на метод `'logging'` в проектах, где такая библиотека логирования не используется);
5. Ложное определение дублирования с последующими рекомендациями по «оптимизации повторяющихся вызовов»;
6. Генерация комментариев с неявной или неочевидной мотивацией, затрудняющей понимание разработчиком цели предложенных изменений;
7. Рекомендации по изменению устоявшихся практик проекта, не подлежащих рефакторингу по соображениям согласованности или усилий.

3.4 Уточнение требований

В работе BitsAI [12] приводится таксономия правил рецензирования кода, в который входят 4 большие группы: Security Vulnerability, Code Defect, Performance Issue и Maintainability and Readability.

В ходе обсуждения внутри команды данные группы были пересмотрены, учитывая исследовательский характер проектов, приводящий к более низким требованиям к надежности и безотказности написанного программного кода. В качестве целевого языка программирования был выбран язык Python. Итоговый список больших групп комментариев, которые хотелось бы видеть от системы автоматического рецензирования, следующий:

- **Programming Best Practices** – применение лучших практик программирования, оценка удобства поддержки, четкости наименований и соответствия PEP 8, с замечаниями по поводу отсутствия документации, только если это затрудняет понимание кода.
- **Bug Fixes** – выявление потенциальных ошибок или улучшение обработки исключений.
- **Optimization** – выявление избыточных или неэффективных вычислений, которые оказывают измеримое влияние.
- **Modern Python Features** – рекомендации по внедрению новых структур и особенностей языка Python.

Каждая из указанных категорий была реализована в качестве опционального поля в рамках структурированного формата вывода, генерируемого моделью.

Дополнительно были интегрированы требования, направленные на предотвращение формирования ложных выводов в ситуациях, требующих межпроцедурного анализа или доступа к внешнему контексту, отсутствующему в текущих входных данных модели. Также были формализованы требования, предписывающие внутреннюю проверку предлагаемых изменений (например, целесообразность устранения предполагаемых избыточных вызовов) и фильтрацию гипотетических рекомендаций, не подкрепленных достаточными основаниями. Вся совокупность данных описанных замечаний и ограничивающих предписаний была интегрирована в системный запрос модели.

Реализация ограничивающих механизмов привела к существенному снижению объема генерируемых замечаний и комментариев. Данное сокращение, при отсутствии способов обогащения контекста, закономерно сужает область применимости автоматизированного рецензента.

3.5 Добавление примеров в инструкцию

Для начала было решено зафиксировать 2 примера с хорошими рецензиями из существующей истории разработки. В первом примере указывалось на использование непонятных идентификаторов имен переменных, слишком общий тип исключений, отсутствие применения библиотечной функции `tempfile` для создания временного файла. Второй пример содержал советы по изменению цикла обработки графа, использованию конструкции `'if cond: return'` для сокращения дальнейшей вложенности кода. Добавление примеров расширило спектр того, о чем пишет модель, но качество новых рецензий оставляло желать лучшего.

В системах генерации ответа с учетом дополнительно найденной релевантной информации (retrieval-augmented generation) [18] выполняется подбор нескольких примеров, наиболее похожих на заданный, по заранее определенной мере, например, косинусному расстоянию между векторными представлениями. В работе [7] для этой задачи применялся подход с подсчетом косинусного расстояния между векторами, полученными из тел функций с помощью метода упрощенного представления текста `CountVectorizer` [19]. Для решения этой задачи был использован подход, описан в работе [7], который был модернизирован

следующим образом: а) замена метода CountVectorizer на схожий подход Bag-of-Words без учета количества слов (токенов) в участке кода, б) замена косинусного расстояния на среднее гармоническое между векторными представлениями, в) использование взвешенной функции SequenceMatcher, которая учитывает больший вклад токенов, попавших в область, выделенную автором для рецензии. Предложенные изменения показали значительный прирост качества определения схожих методов с учетом выделенных участков на закрытом наборе тестов.

Кроме того, важно иметь относительно чистый набор рецензий, которые будут подаваться в качестве примеров [8], такие как "Done" или "что это такое?" могут только ухудшить качество итоговой генерации. Для решения этой задачи использовался классификатор комментариев к исходному коду [20]. В данном исследовании была дообучена языковая модель на базе архитектуры CodeBERT, которая классифицирует рецензии на одну из 5 категорий: Development, Code Style, Discussion, User, Other. Для дальнейшей работы будут использованы только комментарии группы Development.

Полученная система автоматического рецензирования с автоподбором примеров значительно превосходит версию с двумя фиксированными примерами, описанными выше. При этом всё ещё остаются ситуации избыточности предложений, сомнительных улучшений, которые в конкретном контексте не нужны или неприменимы, лишние проверки, попытки исправить несуществующий ошибку.

3.6 Дополнительная фильтрация

В работе [10] также описана инструкция для автоматической оценки качества сгенерированных рецензий. Выбранный автором подход носит название "большая языковая модель как судья" – LLM-as-a-Judge, он представляет собой метод автоматической оценки текстовых ответов, сгенерированных большими языковыми моделями, с использованием другой языковой модели в качестве судьи. Этот подход позволяет имитировать человеческую оценку, фокусируясь на заданных критериях качества, таких как релевантность, точность, стиль или соответствие контексту [21]. В исследовании [22] авторы систематизировали метод, продемонстрировав, что LLM-судьи (например, GPT-4) достигают согласия с человеческими оценками в 85% случаев, что выше, чем уровень согласия между людьми (81%). Выделяют 3 подхода к оценке:

- Попарное сравнение (Pairwise Comparison): LLM выбирает лучший ответ из двух вариантов, например, для сравнения разных версий модели или запросов.
- Прямая оценка (Direct Scoring): LLM присваивает баллы или метки (например, 1–5) на основе критериев: корректность, отсутствие предвзятости, соответствие тону.
- Оценка с контекстом (Reference-Based): LLM проверяет соответствие ответа исходному документу, что полезно для выявления галлюцинаций.

Примером прямой оценки является такой запрос: "Оцените ответ чат-бота по шкале от 1 до 5, где 1 – полностью нерелевантный, 5 – идеально соответствует вопросу. Объяснения не требуются".

Почему это вообще работает, ведь модель не изменилась и какого-то дополнительного контекста, и новых знаний модели подать нельзя, иначе логичнее было бы его передать сразу в задачу генерации? Основным аргументом является то, что задача классификации (например, "содержит ли текст предвзятость?") требует меньших вычислительных ресурсов и проще для LLM, чем генерация связанного ответа.

В работе [10] используются два варианта подхода "модель как судья", первый из них – это прямая оценка по 10 категориям, которые авторы выделили и для самой генерации рецензий. По каждой из этих категорий модели предлагается выставить оценку от 1 до 10 и сделать по

ним итоговый вывод. Второй вариант оценщика, предложенный авторами этой работы, это модель, которая ранжирует несколько ответов, основываясь на все тех же 10 категориях.

Применение представленной инструкции для использования на своих данных показало очень низкий коэффициент согласия с ручной разметкой (коэффициент каппа Коэна меньше 0.01). Тем не менее даже такой запрос отсеивал больше плохих/бесполезных комментариев, чем хороших. Отсюда возникло желание улучшить инструкцию этого инструмента, но использовать его как дополнительную проверку предложений модели и фильтрации ошибочных рецензий. Для этого был проведен ряд экспериментов по улучшению запроса.

Первоначально, проведённый анализ признаков оценки привёл к сокращению исходного набора из 10 метрик до 5 ключевых категорий: Understanding (Понимание), Relevance (Релевантность), Actionability (Выполнимость), Accuracy (Правильность), Clarity (Ясность). Данное решение было обусловлено тем, что исключённые признаки оказались не столь важными и скорее отсеивающими очень плохие по структуре комментарии, которые системы автоматического рецензирования на базе языковой модели пишут редко, при этом их наличие вносило свой положительный вклад в некачественные рецензии и негативно влияло на точность оценки общего качества.

Последующим шагом стал переход от 10-балльной оценочной шкалы к трехуровневой категоризации (poor, medium, good). Это изменение было мотивировано высокой субъективной сложностью для аннотаторов в дифференциации близких численных оценок (например, разграничение баллов 5, 6, 7 по метрике Actionability) и эмпирически подтверждённым превосходством LLM в классификации текстовых меток над числовыми прогнозами. Как демонстрируется в работе [22], использование семантически прозрачных текстовых дескрипторов повышает точность классификации LLM на 15–20%.

Далее, в ходе итеративного процесса валидации (включающего анализ ошибок модели на репрезентативных примерах) были сформулированы 5 групп нормативных требований к качеству комментариев и уточнены правила агрегации частных оценок в итоговую категорию.

Наконец, для обеспечения контекстного понимания, в системный запрос были интегрированы 3 примера (фрагменты кода с сопутствующими рецензиями), сопровождаемые развёрнутыми пояснениями применения установленных правил оценки к данным случаям. Это обеспечило явную демонстрацию ожидаемого формата и логики оценки модели.

Согласованность между моделью с итоговым запросом и ручной разметкой по метрике каппа Коэна достигла 0.14, что всё ещё является недостаточным для использования подхода на практике, но значительно выше первоначального. Основной его пользой для практического применения является пост-фильтрация рецензий. Для каждого комментария моделью была выставлена оценка: ошибочные, средние и хорошие (wrong, medium, good). Далее возможны два варианта работы: **мягкая** – когда убираются только ошибочные комментарии и, соответственно, остаются средние и хорошие; **жесткая** – когда остаются только хорошие комментарии, а все остальные отфильтровываются.

Все использованные и разработанные инструкции опубликованы и доступны по в репозитории [23].

4. Результаты

На рис. 1 приведены примеры рецензий и ручной разметки, которую проводили эксперты. Применимые комментарии говорят о явных проблемах, таких как отсутствие контекстного менеджера для корректной обработки закрытия файла или о неэффективной обработке элементов файла, которые можно сократить использованием генератора списка. К рецензиям среднего качества можно отнести сообщение о некорректном именовании переменных или о несогласованном формате чтения файла и указанном типе данных, что так же может быть

опечаткой и не приведет к значительным нарушениям в работе программы. Такие замечания могут оказаться полезными, либо натолкнуть на мысль о необходимости внести изменения в данный участок кода. К ошибочным комментариям относятся просьбы исправить код на тождественный существующему или желание внедрить в программу библиотеку записи логов, хотя в данном контексте это абсолютно не нужно.

Пример кода

```
import os

def process_user_data(user_id: int, file_name: str, other_line: str) -> None:
    data_efficasy = 0

    full_path = os.path.join('data/', file_name)
    file: IO[str] = open(full_path, 'rb')
    for line in file:
        if line == other_line:
            data_efficasy += 1

    print(f"Data processed with efficasy: {data_efficasy}")
```

Применимые комментарии

```
Line: 7
The file should be opened using a context manager to ensure it gets closed properly.
Suggested fix: with open(full_path, 'rb') as file:

Line: 8
The loop might be transformed to sum function with generator comprehension expression.
Suggested fix: data_efficasy = sum(1 for line in file if if line == other_line)
```

Средние комментарии

```
Line: 2
Variable name 'data_efficasy' is misspelled. It should be 'data_efficiency'.
Suggested fix: data_efficiency = 0

Line: 7
The file is opened in binary mode ('rb') but read as text. Change the mode to 'r' for text
files.
Suggested fix: file: IO[str] = open(full_path, 'r')
```

Ошибочные комментарии

```
Line: 1
The function should import the required module 'os' at the beginning of the file.
Suggested fix: import os

Line: 12
Avoid using print for logging. Use a logging library like logging for better control and
flexibility.
Suggested fix: import logging; logging.basicConfig(level=logging.INFO); logging.info(f"Data
processed with efficasy: {data_efficasy}")
```

Рис. 1. Примеры рецензий и их ручной разметки.
Fig. 1. Examples of review and its manual labels.

Рис. 2 демонстрирует репрезентативный пример кода и соответствующих рецензий, сгенерированных моделью при использовании различных промптов, описанных в Разделе 3. Ключевым замечанием к анализируемой функции от настоящих рецензентов является повторная компиляция регулярных выражений при каждом вызове метода, что создает избыточные вычислительные затраты. С точки зрения оптимизации производительности, предпочтительным решением было бы вынесение предварительно скомпилированных выражений в область глобальных переменных. Рецензия, сформированная на основе базового запроса, содержит ошибочное утверждение о недостаточной выразительности регулярных выражений и их неспособности корректно обрабатывать целевые значения.

Пример кода

```
def get_compiler(config_file_path: Optional[Path]) -> str:
    default_compiler = 'gcc-10'

    if config_file_path is not None and config_file_path.exists():
        with open(config_file_path, 'r', encoding='utf-8') as f:
            config = f.read()
            gcc_pattern = re.compile(r'CONFIG_CC_VERSION_TEXT=+.+gcc.*?(\\d+)')
            clang_pattern = re.compile(
                r'CONFIG_CC_VERSION_TEXT=+.+clang version (\\d+)')
        )

        gcc_match = gcc_pattern.search(config)
        if gcc_match:
            version = gcc_match.group(1)
            return f'gcc-{{version}}'

        clang_match = clang_pattern.search(config)
        if clang_match:
            version = clang_match.group(1)
            return f'clang-{{version}}'

    return default_compiler
```

Базовая инструкция

The regular expressions used to extract the compiler version are too simplistic and may not handle all valid version formats. For example, GCC and Clang versions can include multiple numbers (e.g., gcc-10.2.0 or clang version 12.0.0). Update the regular expressions to capture the full version string. Additionally, consider adding error handling for cases where the version string is not found or is malformed.

Расширенный контекст

The regex patterns are compiled inside the function, which means they are recompiled every time the function is called. Move the pattern compilation outside the function to improve performance.

Suggested fix:

```
gcc_pattern = re.compile(r'CONFIG_CC_VERSION_TEXT=+.+gcc.*?(\\d+)')
clang_pattern = re.compile(r'CONFIG_CC_VERSION_TEXT=+.+clang version(\\d+)')
def get_compiler(config_file_path: Optional[Path]) -> str:"
```

Улучшение требований

The gcc_pattern and clang_pattern are compiled every time the function is called. These patterns are static and can be compiled once outside the function to improve performance.

Suggested fix:

```
gcc_pattern = re.compile(r'CONFIG_CC_VERSION_TEXT=+.+gcc.*?(\\d+)')
clang_pattern = re.compile(r'CONFIG_CC_VERSION_TEXT=+.+clang version (\\d+)')
def get_compiler(config_file_path: Optional[Path]) -> str:
    default_compiler = 'gcc-10'
    if config_file_path is not None and config_file_path.exists():
        with open(config_file_path, 'r', encoding='utf-8') as f:
            config = f.read()
            gcc_match = gcc_pattern.search(config)
            if gcc_match:
                version = gcc_match.group(1)
                return f'gcc-{{version}}'
            clang_match = clang_pattern.search(config)
            if clang_match:
                version = clang_match.group(1)
                return f'clang-{{version}}'
    return default_compiler
```

Автоподбор примеров

Avoid reading the entire file into memory by using Path.read_text() for better readability and performance.

Suggested fix:

```
config = config_file_path.read_text()
```

Рис. 2. Примеры рецензий при различных инструкциях.
Fig. 2. Examples of review from different prompts.

Данное заключение не соответствует действительности, так как текущая реализация метода обеспечивает полное покрытие требуемых случаев. Запрос с расширенным контекстом, напротив, корректно идентифицирует проблему повторной компиляции регулярных выражений. В предложенном исправлении модель выносит регулярные выражения за его пределы, однако вносит необоснованные изменения в сами выражения.

Усовершенствованный вариант запроса генерирует рецензию, не только согласующуюся с предыдущей инструкцией, но и предлагает полный исправленный метод с использованием вынесенных глобальных переменных, сохраняя при этом исходные регулярные выражения без изменений. Наконец, рецензия, полученная при использовании запроса с автоматически подобранными примерами содержит новое замечание о том, что вместо самостоятельного чтения файла через функцию `open` можно использовать встроенный метод `read_text()`, так как переменная `config_file_path` является объектом типа `Path` из библиотеки `pathlib`.

Для численной оценки применимости предложенных рецензий был использовано два тестовых стенда. Первый был составлен из закрытого набора 123 добавленных и измененных методов, извлеченных из 10 коммитов локального сервера Gerrit. Корпус включает 11 новых и 25 модифицированных файлов, реализованных на языке Python. Аннотирование данных выполнялось двумя авторами исходного кода с целью точной классификации предложений, сгенерированных LLM, по следующим категориям: применимые, приемлемые (средние), но несущественные в данном контексте, и ошибочные, с низкой вероятностью внедрения разработчиком. Согласованность при ручной разметке, измеренная каппой Коэна, составила 0.81. В качестве базовой модели использовалась большая языковая модель Qwen/Qwen2.5-Coder-32B-Instruct. Среднее время обработки одного набора изменений составило примерно 75 секунд, а среднее время дополнительной оценки одной рецензии около 7 секунд.

В табл. 1 приведены результаты разметки сгенерированных рецензий до применения фильтров. Наилучшие результаты показывает подход с улучшенными требованиями в инструкции как по точности (28.89%), так и по количеству применимых (26 против 22 у базовой инструкции). Стоит отметить, что автоматический подбор примеров повышает количество и точность неплохих (средних и применимых) рецензий (50.82%). Также интересным наблюдением является достаточно низкие показатели у подхода с фиксированными примерами, точность применимых замечаний даже ниже, чем у базовой инструкции. Таким образом применение вышеописанных методов улучшения инструкции повысило точность полностью применимых рецензий в 2 раза, а неплохих примерно в 1.4 раза. В результатах тестирования работы BitsAI на модели Qwen2.5-Coder-32B-instruct указана средняя точность полезных комментариев на уровне 10%, что немного ниже, но сравнимо с полученным в табл. 1 для базовой инструкции.

В табл. 2 и 3 приведены результаты применения инструкции из секции 3.6 в качестве мягкой и жесткой пост-фильтрации сгенерированных рецензий, соответственно. По метрике точности жесткая фильтрация показывает более высокие (до 37% применимых) результаты, однако общее количество, в том числе и неплохих комментариев сильно снижается (40 против 52 для подхода с автоподбором примеров). Из таблицы видно, что применение одной только фильтрации к базовой инструкции не дает такого прироста точности, как настройка инструкции, однако в сумме с остальными подходами значительно повышает итоговую точность (в 2.5 раза применимые до 37% и в 1.6 раза неплохие рецензии до 61%). Также подход с мягкой и жесткой фильтрациями добавляет гибкость в настройку итогового инструмента.

Второй тестовый набор состоял из 1000 экземпляров открытого набора данных с примерами изменения кода и соответствующих им рецензий CRer, представленного в работе [10]. Исследуемая модель (инструкция с улучшенными требованиями) сгенерировала рецензии для представленных фрагментов кода. Относительная релевантность выводов оценивалась с помощью ранжирующей системы DeepCRCEval, сортирующей комментарии по степени соответствия целевому коду.

Табл. 1. Результаты оценки качества рецензий (без фильтров).

Table 1. Results of the reviews evaluation (without filters).

Метод	Общее количество	Ошибочные	Средние (шт, %)	Применимые (шт, %)
Базовая инструкция	150	95	33, 22	22, 14.7
Расширенный контекст	123	74	27, 21.9	22, 17.9
Улучшение требований	90	48	16, 17.8	26, 28.89
Фиксированные примеры	150	92	37, 24.6	21, 14
Автоподбор примеров	122	60	38, 31.1	24, 19.7

Табл. 2. Результаты оценки качества рецензий после мягкой фильтрации.

Table 2. Results of the reviews evaluation after soft filtering.

Метод	Общее количество	Ошибочные	Средние (шт, %)	Применимые (шт, %)
Базовая инструкция	106	61	26, 24.52	19, 17.9
Расширенный контекст	80	42	21, 26.25	17, 21.25
Улучшение требований	64	29	14, 21.87	21, 32.81
Фиксированные примеры	78	34	27, 34.6	17, 21.79
Автоподбор примеров	92	40	31, 33.9	21, 22.83

Табл. 3. Результаты оценки качества рецензий после жесткой фильтрации.

Table 3. Results of the reviews evaluation after hard filtering.

Метод	Общее количество	Ошибочные	Средние (шт, %)	Применимые (шт, %)
Базовая инструкция	76	42	19, 25	15, 19.74
Расширенный контекст	51	22	14, 27.4	15, 29.41
Улучшение требований	54	21	13, 24.07	20, 37.04
Фиксированные примеры	47	24	12, 25.53	11, 23.4
Автоподбор примеров	66	26	22, 33.3	18, 27.27

Данный оценщик осуществляет сортировку списка рецензий по степени их соответствия (релевантности) целевому поданному участку кода. В каждый ранжируемый пакет для одного фрагмента кода включались N комментариев, сгенерированных усовершенствованной моделью, и один комментарий, полученный с использованием промпта из системы DeepCRCEval. Ключевое различие заключается в формате вывода: базовый подход генерирует единое сообщение на фрагмент кода, тогда как усовершенствованная инструкция пишет список отдельных замечаний. Для обеспечения сопоставимости результатов при вариативном значении N , рецензии, сгенерированные базовым запросом к системе DeepCRCEval, которым оценщик присваивал позицию ниже первой, принудительно устанавливались на ранг 2. Эта мера предотвратила искажение метрики средне обратного ранга (Mean Reciprocal Rank, MRR) из-за не фиксированного количества конкурирующих рецензий от усовершенствованной модели в разных пакетах. Усовершенствованный подход продемонстрировал значимое превосходство, достигнув среднего ранга (MRR) 0.915, в то время как результат базовой инструкции составил 0.585. Полученные значения средне обратного ранга свидетельствуют о более высокой эффективности разработанного подхода в генерации применимых рецензий.

5. Заключение

В рамках данного исследования была осуществлена разработка и детальная проработка специализированных инструкций, направленных на эффективное решение комплексной задачи автоматизированной генерации содержательных и применимых рецензий для изменений исходного кода. Для достижения этой цели был проведен анализ и последующее практическое внедрение ряда современных подходов к конструированию промптов, доказавших свою эффективность в смежных областях обработки естественного языка и кода. К числу этих использованных методик относятся: пошаговые рассуждения, предоставление модели ограниченного количества релевантных примеров, структурированный вывод и интеллектуальная фильтрация отдельной инструкцией оценщика качества. Для проведения экспериментов была выбрана открытая модель Qwen/Qwen2.5-Coder-32B-Instruct. Проведена оценка применимости предложенных рецензий непосредственными авторами изменений. Наилучший из реализованных и протестированных подходов к составлению инструкций продемонстрировал значительный качественный скачок: доля применимых рецензий увеличилась в 2.5 раза, одновременно повысив процент неплохих комментариев в 1.6 раза, что позволило достичь точности в 37% и 61% соответственно. Анализ экспериментальных данных позволил выявить, что предоставление модели небольшого количества примеров повышает общее количество неплохих комментариев. Кроме того, было продемонстрировано, что использование отдельного ИИ эксперта-оценщика, может быть эффективно в качестве интеллектуального ранжирующего фильтра для рецензий. Основная цель такого фильтра – повышение точности и количества полезных комментариев в финальном выводе системы при одновременном отсеивании нерелевантных, избыточных или потенциально вводящих в заблуждение предложений, тем самым существенно повышая итоговую ценность и пользовательский опыт автоматизированного рецензирования для конечного пользователя.

Список литературы / References

- [1]. Siow, J. K., Gao, C., Fan, L., Chen, S., & Liu, Y. (2020). Core: Automating review recommendation for code changes. 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 284-295.
- [2]. Tufano, R., Pasarella, L., Tufano, M., Poshyvanyk, D., & Bavota, G. (2021). Towards automating code review activities. In 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 163-174.

- [3]. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140), 2020, pp. 1-67.
- [4]. Tufano, R., Masiero, S., Mastropaolo, A., Pascarella, L., Poshyvanyk, D., Bavota, G. (2022). Using pre-trained models to boost code review automation. *Proceedings of the 44th international conference on software engineering* pp. 2291-2302.
- [5]. Li, Z., Lu, S., Guo, D., Duan, N., Jannu, S., Jenks, G., Sundaresan, N. (2022). Automating code review activities by large-scale pre-training. *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1035-1047.
- [6]. Li, L., Yang, L., Jiang, H., Yan, J., Luo, T., Hua, Z., Zuo, C. (2022). AUGER: automatically generating review comments with pre-training models. *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1009-1021.
- [7]. Hong, Y., Tantithamthavorn, C., Thongtanunam, P., Aleti, A. (2022). Commentfinder: a simpler, faster, more accurate code review comments recommendation. *Proceedings of the 30th ACM joint European software engineering conference and symposium on the foundations of software engineering*, pp. 507-519.
- [8]. Liu, C., Lin, H. Y., Thongtanunam, P. (2025). Too noisy to learn: Enhancing data quality for code review comment generation. In *2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR)*, pp. 236-248.
- [9]. Yu, Y., Zhang, L., Rong, G., Shen, H., Zhang, J., Yan, H., Tian, Z. (2024). Distilling Desired Comments for Enhanced Code Review with Large Language Models. Доступно по ссылке: <https://arxiv.org/abs/2412.20340>.
- [10]. Lu, J., Li, X., Hua, Z., Yu, L., Cheng, S., Yang, L., Zuo, C. (2025). Deepcrceval: Revisiting the evaluation of code review comment generation. *International Conference on Fundamental Approaches to Software Engineering*, pp. 43-64.
- [11]. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, vol. 35, pp. 24824-24837.
- [12]. Sun, T., Xu, J., Li, Y., Yan, Z., Zhang, G., Xie, L., Sui, K. (2025). BitsAI-CR: Automated Code Review via LLM in Practice. Доступно по ссылке: <https://arxiv.org/abs/2501.15134>.
- [13]. Hui B, Yang J, Cui Z, Yang J, Liu D, Zhang L, Liu T, Zhang J, Yu B, Lu K, Dang K. (2024). Qwen2.5-coder technical report. arXiv preprint arXiv:2409.12186.
- [14]. Structured outputs: Everything you should know, Доступно по ссылке: <https://humanloop.com/blog/structured-outputs>, дата обращения 10.06.2025.
- [15]. Structured outputs in llms: Definition, techniques, applications, benefits, Доступно по ссылке: <https://www.leewayhertz.com/structured-outputs-in-llms/>, дата обращения 10.06.2025.
- [16]. How to return structured data from a model, Доступно по ссылке: https://python.langchain.com/docs/how_to/structured_output/, дата обращения 10.06.2025.
- [17]. Pydantic, Доступно по ссылке: <https://docs.pydantic.dev/latest/>, дата обращения 10.06.2025.
- [18]. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, vol. 33, pp. 9459-9474.
- [19]. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *The Journal of machine Learning research*, vol. 12, pp. 2825- 2830.
- [20]. Петрова П.А., Марков С.И., Качанов В.В. Создание набора данных для комбинированной классификации рецензий исходного кода. Интеллектуализация обработки информации. Тезисы докладов 15-й международной конференции, 2024г., стр. 83-85.
- [21]. Llm-as-a-judge: a complete guide to using llms for evaluations, Доступно по ссылке: <https://www.evidentlyai.com/llm-guide/llm-as-a-judge>, дата обращения 10.06.2025.
- [22]. Zheng, L., Chiang, W. L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Stoica, I. (2023). Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, vol. 36, pp. 46595-46623.
- [23]. LLM review prompts, Доступно по ссылке: https://github.com/vova98/llm_review_prompts, дата обращения 10.06.2025.

Информация об авторах / Information about authors

Владимир Владимирович КАЧАНОВ – аспирант кафедры интеллектуальных систем МФТИ (НИУ), программист отдела системного программирования Института системного программирования с 2021 года. Сфера научных интересов: программная инженерия, машинное обучение, обработка естественного языка.

Vladimir Vladimirovich KACHANOV – postgraduate student of the Department of Intelligent Systems at MIPT (National Research University), programmer of the Department of Systems Programming at the Institute of Systems Programming since 2021. Research interests: software engineering, machine learning, natural language processing.

DOI: 10.15514/ISPRAS-2025-37(4)-23



Извлечение знаний в ограниченной области для примеров состязательных атак «черного ящика»

^{1, 2, 3} К.С. Лукьянов, ORCID: 0009-0009-5235-2175 <lukianov@ispras.ru>

¹ А.И. Перминов, ORCID: 0000-0001-8047-0114 <perminov@ispras.ru>

^{1, 3} Д.Ю. Турдаков, ORCID: 0000-0001-8745-0984 <turdakov@ispras.ru>

^{3, 4} М.А. Паутов, ORCID: 0000-0003-0438-6361 <pautov@airi.net>

¹ Институт системного программирования РАН,
Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.

² Московский физико-технический институт (НИУ),
Россия 117303, Москва, ул. Керченская, д.1 А, корп. 1.

³ Исследовательский центр доверенного искусственного интеллекта ИСП РАН,
Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.

⁴ AIRI, Россия 105064, Москва, р. 19, Нижний Сусальный пер., 5

Аннотация. Устойчивость нейронных сетей к состязательным возмущениям в условиях «чёрного ящика» остаётся сложной проблемой. Большинство существующих методов атак требуют чрезмерного количества запросов к целевой модели, что ограничивает их практическую применимость. В данной работе мы предлагаем подход, в котором суррогатная модель-ученик итеративно обучается на неудачных попытках атак, постепенно изучая локальное поведение модели «чёрного ящика». Эксперименты показывают, что этот метод значительно сокращает количество необходимых запросов, сохраняя при этом высокую вероятность успеха атак.

Ключевые слова: состязательная атака черного ящика; извлечение знаний.

Для цитирования: Лукьянов К.С., Перминов А.И., Турдаков Д.Ю., Паутов М.А. Извлечение знаний в ограниченной области для примеров состязательных атак «черного ящика» Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 133–146. DOI: 10.15514/ISPRAS-2025-37(4)-23.

Благодарности: Работа поддержана грантом, предоставленным Министерством экономического развития Российской Федерации в соответствии с Соглашением о предоставлении из федерального бюджета гранта в форме субсидии федеральному государственному бюджетному учреждению науки Институт системного программирования им. В.П. Иванникова Российской академии наук от 20 июня 2025 г. № 139-15-2025-011. Идентификатор соглашения 000000Ц313925P4G0002.

Knowledge Distillation in Local-Region for Black-Box Adversarial Examples

^{1, 2, 3} K.S. Lukianov ORCID: 0009-0009-5235-2175 <lukianov@ispras.ru>

¹ A.I. Perminov, ORCID: 0000-0001-8047-0114 <perminov@ispras.ru>

^{1, 3} D.Y. Turdakov, ORCID: 0000-0001-8745-0984 <turdakov@ispras.ru>

^{3, 4} M.A. Pautov, ORCID: 0000-0003-0438-6361 <pautov@airi.net>

¹ Institute of System Programming of the Russian Academy of Sciences,
25, A. Solzhenitsyn str., Moscow, 109004, Russia.

² Moscow Institute of Physics and Technology (National Research University),
building 1, 1 A, Kerchenskaya str., Moscow, 117303, Russia.

³ Research Center for Trusted Artificial Intelligence ISP RAS,
25, A. Solzhenitsyn str., Moscow, 109004, Russia.

⁴ AIRI, 5, Nizhniy Susalny per., p. 19, Moscow, 105064, Russia.

Abstract. The robustness of neural networks to adversarial perturbations in black-box settings remains a challenging problem. Most existing attack methods require an excessive number of queries to the target model, limiting their practical applicability. In this work, we propose an approach in which a surrogate student model is iteratively trained on failed attack attempts, gradually learning the local behavior of the black-box model. Experiments show that this method significantly reduces the number of queries required while maintaining a high attack success rate.

Keywords: black-box adversarial attack; knowledge distillation.

For citation: Lukianov K.S., Perminov A.I., Turdakov D.Y., Pautov M.A. Knowledge Distillation in Local-Region for Black-Box Adversarial Examples Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025, pp. 133-146 (in Russian). DOI: 10.15514/ISPRAS-2025-37(4)-23.

1. Введение

Устойчивость глубоких нейронных сетей к возмущениям входных данных является ключевым свойством, необходимым для их интеграции в различные области машинного обучения с повышенными требованиями к безопасности. Например, в прикладных задачах: автономного вождения, медицинской диагностики и финансовых технологиях вопросы безопасности имеют очень высокое значение. Хотя от нейронных сетей ожидается, что они будут выдавать схожий результат для близких по содержанию входов, давно известно, что они уязвимы к так называемым состязательным (*adversarial*) возмущениям [1], малым, специально сконструированным изменениям входных данных, которые не меняют семантику исходного объекта, но вынуждают модель выдавать заранее определённый результат. Большинство методов генерации состязательных атак нейронных сетей направлено на построение подобных возмущений, что демонстрирует, что в целом прогнозы нейронных сетей нельзя считать полностью надёжными.

Наиболее эффективные и незаметные атаки, как правило, требуют доступа к градиентам модели и поэтому имеют ограниченную практическую применимость [2-4]. Однако в реальных условиях модели машинного обучения нередко развёртываются в виде сервисов, доступных через прикладной интерфейс (*Application Programming Interface, API*). Такой режим работы, хотя и накладывает определённые ограничения на исследование робастности моделей класса MLaaS (*Machine Learning as a Service*), не исключает возможности генерации *состязательных* возмущений [5-8]. В частности, для «чёрного ящика» (когда доступа есть только к ответам модели, а не ко всей модели, в частности к ее весам) можно построить возмущение, оценивая градиенты в окрестности целевой точки [9-10], применяя методы случайного поиска [6] или используя перенос знаний для обучения вспомогательной модели

и проведения атаки в режиме «белого ящика» (когда у атакующего есть полный доступ к модели, включая ее веса и градиенты) [11-12].

Тем не менее, методы черного ящика зачастую требуют значительного числа запросов к целевой модели. В настоящей работе мы ставим следующий исследовательский вопрос: **как сократить число необходимых запросов для построения *состязательных* примеров для заданной модели черного ящика, решающей задачу классификации изображений с сотен и тысяч до десятков запросов?** Для ответа на него мы предлагаем метод *Mimic* — метод генерации атаки на черный ящик. В основе алгоритма находится идея переноса атаки, полученной на модели-заместителе (далее модель-ученик), построенной на итеративной локализованной дистилляции знаний, на модель черного ящика.

Методы атак, основанные на дистилляции знаний, в последние годы активно изучаются. Они применяются, например, для защиты интеллектуальной собственности: полученная в результате извлечения знаний моделью учеником используется для внедрения цифровых водяных знаков, позволяющих установить источник сгенерированного контента [13-16]. Данный подход также используется при атаках на модели черного ящика [11-12]. В рамках нашей работы мы предлагаем итеративное обучение серии моделей-учеников на расширяющемся наборе данных. Такой процесс позволяет каждой последующей модели ученику точнее воспроизводить поведение исходной модели черного ящика по сравнению с предыдущей и, в конечном счёте, вычислять *состязательное* возмущение для неё.

Наш вклад можно представить следующим образом:

1. Мы предлагаем методику *Mimic* – атаку с переносом на модель черного ящика, основанную на локальной дистилляции знаний. Алгоритм использует поведение целевой (учительской) модели в окрестности рассматриваемой точки в пространстве признаков и формирует набор моделей-учеников, воспроизводящих предсказания целевой модели на конечном множестве точек. Далее этот набор моделей используется для вычисления *состязательных* возмущений в режиме «белого ящика», которое после конечного числа итераций дистилляции переносится на модель-учителя.
2. Мы экспериментально демонстрируем эффективность предлагаемого подхода по сравнению с другими методами атаками черного ящика в задаче классификации изображений.

2. Обзор литературы

В данном разделе приводится краткий обзор существующих атак черного ящика и применений дистилляции знаний.

2.1 Создание состязательных возмущений

В литературе методы построения состязательных возмущений для сетей с ограниченным доступом к архитектуре, весам или обучающему набору данных называются атаками в режиме «чёрного ящика». Такие атаки подразделяются на два типа: запросно-ориентированные (*query-based*) и переносные (*transfer-based*). В данной работе рассматривается переносимость состязательных атак из режима «белого ящика» в режим «чёрного ящика».

В запросно-ориентированных атаках злоумышленник использует выход целевой модели для построения состязательного примера. Один из подходов — оценка градиента модели по входным данным [17-20]. Однако подобные методы, как правило, требуют большого числа запросов к целевой модели, что делает их непрактичными. В противоположность им, при переносных атаках состязательный пример формируется путём атаки на одну или несколько вспомогательных (замещающих) моделей [7, 11, 21]. В литературе описаны подходы, улучшающие переносимость, включая использование аугментаций данных [22], методов

дополненного использованием градиентов [23], агрегации [24] или настройки направлений поиска [25].

Среди известных атак чёрного ящика можно выделить, например, ZOO [18] и NES [9]. Атака ZOO последовательно добавляет небольшие положительные или отрицательные возмущения к каждому пикселю целевого изображения, после чего отправляет измененные изображения на вход черного ящика и, опираясь на ответы модели, производит оценку градиента в окрестности исходного изображения. Атака NES действует аналогично, но вместо последовательного изменения пикселей использует набор случайно сгенерированных изображений для оценки градиента модели.

Существуют два варианта постановки задачи атаки на классификационные модели в режиме «чёрного ящика» в зависимости от выходных данных целевой модели: атака с мягкими метками (*soft-label*, когда на выходе вектор логитов, то есть полный вектор сделанных моделью оценок принадлежности входных данных каждому возможному классу) и атака с жёсткими метками (*hard-label*, когда на выходе индекс наиболее вероятного класса, по оценке модели).

На сегодняшний день среди запросно-ориентированных атак наилучшие показатели по среднему количеству запросов демонстрируют атаки Square Attack [6], NP-Attack [10] и MCG [26] (в случае мягких меток), а также GeoDA [27], Rays [28] и PSBA [29] (в случае жёстких меток). Среди переносных атак одной из наиболее эффективных считается Bayesian attack [11].

Square Attack выбирает область изображения для модификации и постепенно изменяет её в ходе работы алгоритма, случайным образом изменяя пиксели внутри этой области. **NP-Attack** использует нейросетевой предсказатель для направления поиска состязательных возмущений, прогнозируя выходные значения модели. **MCG** представляет собой мета-обучаемую атаку чёрного ящика, в которой мета-классификатор обобщает состязательные атаки на различные модели черного ящика.

Метод **GeoDA** рассматривает состязательную атаку как задачу геометрического исследования, фокусируясь на локальных свойствах границы решений. Алгоритм итеративно изменяет вход, получая жёсткие метки от модели, и уточняет границу между двумя классами для построения атаки. Метод **Rays** использует стратегию поиска на основе принципа трассировки лучей, постепенно продвигаясь в пространстве входных данных до пересечения с границей решений. Атака на чёрный ящик с прогрессивной шкалой границ **PSBA** (*Progressive-Scale Boundary Blackbox Attack*) применяет проективную оценку градиента по жёстким меткам, постепенно корректируя масштаб возмущений.

Атака **Bayesian** повышает переносимость состязательных примеров с помощью замещающей модели с байесовскими свойствами, используя стохастические веса и метод Монте-Карло-дропаута (*Monte Carlo Dropout*) для улучшенной оценки неопределённости. Этот метод относится к переносным атакам и предполагает наличие доступа к части обучающих данных, на которых обучалась модель черного ящика, либо к другому набору данных с аналогичным распределением. В нашей работе предполагается, что злоумышленник не имеет доступа к обучающим данным, поэтому мы не сравниваем наш подход с методами, требующими такого доступа.

2.2 Дистилляция знаний и защита от состязательных возмущений

Дистилляция знаний (*Knowledge Distillation*, KD) – это метод передачи качества работы большой модели-учителя на меньшую, облегчённую модель-ученика [30]. Пусть задана модель-учитель **T**. Обучение модели-ученика **S** из параметрического семейства моделей **S'** формулируется как задача оптимизации:

$$S = \operatorname{argmin}_{S'} \int_{D} [\alpha L(S'(x), y) + (1 - \alpha) \tau^2 \operatorname{KL}(S'(x), T(x))] dx$$

где D – набор данных для дистилляции, L – функция потерь для задачи классификации, KL – дивергенция Кульбака–Лейблера, α и τ – скалярные параметры.

Метод дистилляции знаний применяется в широком спектре задач: сжатие моделей [31-33], обеспечение приватности данных [16, 34-36], адаптация для больших языковых моделей [37-39] и диффузионных моделей [40-42].

В последние годы показано, что дистилляция знаний может повышать состязательную устойчивость к аддитивным возмущениям [43-45]. В отличие от больших моделей-учителей, которые могут достигать высокого уровня робастности, для небольших моделей-учеников сложно одновременно обеспечить высокую устойчивость и сохранение точности [45]. Для решения этой проблемы была предложена состязательно-устойчивая дистилляция, при которой в процессе дистилляции учитываются предсказания на чистых данных [46] или вероятностные векторы [47] робастной модели-учителя.

3. Постановка задач

В этом разделе формально вводится постановка задач, вводятся обозначения, используемые в статье, и формулируются вопросы исследования. Введем формальные определения атак уклонения, отравления, защищенных моделей от соответствующих атак и частично защищенной модели.

3.1 Атаки на модель классификации

Атака уклонения (*evasion attack*) для моделей классификации – это процесс, при котором злоумышленник стремится изменить входные данные $x \in R^d$ таким образом, чтобы модифицированные данные $x' = x + \delta$ изменили прогноз модели на другой класс $f: R^d \rightarrow Y$, где Y – множество классов.

Целью атаки уклонения является нахождение возмущения $\delta \in R^d$, удовлетворяющего следующим условиям:

1. $\operatorname{argmax}_{y \in Y} f(x') \neq \operatorname{argmax}_{y \in Y} f(x)$, то есть предсказанный класс изменяется после применения возмущения.
2. $\|\delta\|_p \leq \epsilon$, где $\|\cdot\|_p$ – L_p норма (например, с $p = 2$ или $p = \infty$), а ϵ – заданное ограничение на величину возмущения, чтобы сохранялась правдоподобность измененного входа.

Таким образом, задача атаки уклонения может быть формализована как задача оптимизации: $\operatorname{argmax}_{y \in Y} f(x') \neq \operatorname{argmax}_{y \in Y} f(x)$, при условии $\|\delta\|_p \leq \epsilon$

Атаки уклонения делятся на два основных типа:

1. Белый ящик (white-box): злоумышленник обладает полным доступом к модели f , включая её параметры и градиенты.
2. Чёрный ящик (black-box): злоумышленник имеет ограниченный доступ к модели f , например, может наблюдать только выходы модели в ответ на определенные входы.

4. Методология

В данном разделе описывается предложенный подход для генерации состязательных примеров для модели-учителя (модель черного ящика) с использованием дистилляции знаний.

4.1 Mimic: ученик следует за учителем

Для выполнения состязательной атаки на модель-учителя (модель черного ящика) T мы сначала применяем процедуру дистилляции знаний и получаем модель-ученика S в режиме «белого ящика». Обучающая выборка для модели-ученика формируется путём запросов к модели-учителю и сохранения её предсказаний для объектов из независимого (hold-out) набора данных D_h , не пересекающегося с обучающей выборкой учителя D .

$$(\mathcal{D}(T) : \mathcal{D}_h \cap \mathcal{D}(T) = \emptyset)$$

В нашей постановке в качестве D_h используется тестовое подмножество набора данных учителя. В зависимости от режима дистилляции,

$$\mathcal{D}(S) = \{(x_i, T(x_i))\}_{i=1}^m \text{ or } \mathcal{D}(S) = \{(x_i, h(T, x_i))\}_{i=1}^m$$

Предполагая, что модель-ученик обладает достаточной обучающей способностью, мы обучаем её до совпадения с моделью-учителем на множестве $D(S)$.

В режиме с мягкими метками модель-ученик должна удовлетворять условиям:

$$\begin{cases} h(S, x_i) = h(T, x_i) = y_i, \\ \|S(x_i) - T(x_i)\|_\infty < \frac{\varepsilon}{4}, \end{cases}$$

где $\varepsilon > 0$ — заданная константа. Второе условие отражает способность модели-ученика уверенно имитировать поведение модели-учителя на $D(S)$.

В режиме с жёсткими метками модель-ученик должна удовлетворять условию:

$$h(S, x_i) = h(T, x_i) = y_i.$$

4.2 Mimic: атака ученика

На этом этапе мы описываем процедуру генерации состязательного примера для модели-ученика.

После обучения модели-ученика мы проводим на ней атаку в режиме «белого ящика», используя метод проецируемого градиентного спуска (*Projected Gradient Descent*, PGD) [48].

Пусть дан входной объект $x \in R^d$ класса $y \in [1, \dots, K]$, который обе модели (учитель и ученик) классифицируют правильно. Метод PGD выполняется итеративно по градиенту для поиска состязательного примера x' в δ -окрестности точки x , то есть в множестве $U_\delta(x) = \{x' : \|x - x'\|_2 \leq \delta\}$.

$$\begin{cases} x^{t+1} = \text{Proj}_{U_\delta(x)} [x^t + \alpha \text{sign} \nabla_{x^t} L(S, x^t, y)], \\ x^1 = x, \\ x' = x^M, \end{cases}$$

где $\alpha > 0$ — величина шага оптимизации, M — число итераций PGD, $\text{Proj}_{U_\delta(x)}$ — оператор проекции в замкнутую δ -окрестность точки x , а $L(S, x_t, y)$ — функция потерь модели S на примере (x_t, y) . В нашей постановке $L(S, x_t, y)$ — это функция кросс-энтропии.

Замечание. Для повышения вычислительной эффективности атаки мы генерируем не один, а пакет из l атакующих примеров $\{x_1', \dots, x_l'\}$ для модели ученика. Псевдокод предлагаемого метода представлен в Алгоритме 1. Обратите внимание, что мы используем атаку PGD, благодаря её простоте; однако наш подход не ограничен конкретным типом атаки белого ящика.

5. Эксперименты

В данном разделе приводится техническое описание проведённых экспериментов. В частности, мы описываем используемые наборы данных и архитектуры моделей, методы базового сравнения, а также методологию проведения экспериментов.

5.1 Условия обучения

В наших экспериментах в качестве обучающих наборов данных для модели-учителя используются CIFAR-10 и CIFAR-100 [49]. В качестве модели-учителя T используется ResNet50 [50], которая обучалась в течение 250 эпох для достижения высокой точности классификации (92% для CIFAR-10 и 75% для CIFAR-100). Для обучения модели-учителя используется оптимизатор SGD с коэффициентом обучения 0.1, коэффициентом регуляризации (weight decay) 10^{-4} и моментумом 0.9.

5.2 Настройка Mimic

В качестве архитектуры модели ученика используется SmallCNN. Мы проводим PGD-атаку на модели-ученике со следующими параметрами: количество шагов PGD $M = 10$, шаг градиента $\alpha = 0.005$, порог расстояния $\delta = 0.05$. В случае мягких меток (soft-label) используется функция потерь L1 Loss, для жёстких меток (hard-label) – CrossEntropyLoss. Архитектура модели SmallCNN и описание алгоритма в виде псевдокода представлены в алгоритме 1. Обращаем внимание, что так как в литературе принято опираться на поиск первого атакующего изображения, то в таком случае необходима досрочная остановка в алгоритме. Алгоритм 1 соответствует варианту, который использовался для получения результатов представленных в таблицах 2-5. Однако, если есть необходимость получить большую серию атак при ограниченном числе запросов, то в алгоритме необходимо убрать досрочные выходы из цикла с помощью операторов выхода из цикла `break`.

5.3 Методы черного ящика для сравнения

В этом подразделе мы кратко перечисляем набор методов, с которыми мы сравниваем наш подход.

В случае мягких меток (soft-label) мы оцениваем MMAAttack по сравнению с алгоритмами ZOO [18] и NES [9] как основными конкурентами. Среди методов атак типа «черного ящика», основанных на случайном поиске, мы выбираем Square attack [6] как современный метод с наименьшим средним числом запросов для проведения атаки. В группе методов, использующих оценку градиента, NP-Attack [10] является одним из наиболее эффективных. В категории комбинированных методов мы выбираем метод MCG [26]. Гиперпараметры методов, использованных в экспериментах, приведены в табл. 1. Отметим, что алгоритм MCG изначально предполагает обучение на данных, распределение которых близко к распределению данных учительской модели, что в общем случае может быть неизвестно. В данном исследовании мы подчеркиваем, что наш метод не имеет такого ограничения.

В случае жестких меток (hard-label) мы оцениваем MMAAttack по сравнению с GeoDA [27] как современным методом с наименьшим средним числом запросов; Rays [28] является одним из наиболее эффективных методов атаки на основе случайного поиска, а PSBA [29] – одним из наиболее эффективных методов в группе атак, использующих оценку градиента.

5.4 Результаты экспериментов

В экспериментах методы ZOO, NES и Square Attack выполнялись 100 раз с различными случайными инициализациями, а методы NP-Attack, MCG и MMA – 30 раз.

В табл. 2 представлен сравнительный анализ существующих методов атак черного ящика атак на наборах данных CIFAR-10 и CIFAR-100 в режиме мягких меток. В таблице приведено

среднее количество запросов (AQN), необходимое для генерации первого атакующего примера для моделей черного ящика и средний успех атаки (ASR), для оценки успешных попыток атаки; символ δ обозначает максимально возможное расстояние до целевой точки в терминах нормы L_2 ; лучшие результаты выделены **жирным шрифтом**. Следует отметить, что, если результаты метода, приведённые в литературе, не совпадают с результатами, полученными в нашей реализации, в таблице приводится результат с наименьшим средним количеством запросов. Результаты, взятые из литературы, отмечены как (Lit).

Данные для метода MCG на наборе CIFAR-100 были взяты из [26]; данные для методов MCG и NP-Attack на наборе CIFAR-10 – из работы [51].

В табл. 3 приведено сравнение существующих методов атак черного ящика атак на наборах данных CIFAR-10 и CIFAR-100 в режиме жестких меток. Данные для метода Rays на CIFAR-10 были взяты из [28]; данные для методов GeoDA на CIFAR-10 – из работы [51].

Input: Черный ящик, который является моделью учителем T , входной объект x класса y , Порог расстояния δ , шаг градиента α , максимальное количество итераций PGD M , максимальное количество итераций дистилляции N , начальный случайный поднабор данных \mathcal{D}_h , количество состязательных примеров l для генерации серии моделей учеников S_2, \dots, S_N , если не будет найдена успешная атака на модель черного ящика.

Output: Набор моделей-учеников $\{S_1, \dots, S_N\}$, набор $AE(T)$ состязательных примеров для модели T .

```

 $z \leftarrow (x, T(x))$  // вычислить логиты модели T в целевой точке
 $\mathcal{D}(S) \leftarrow \{(x_i, T(x_i))\}_{i=1}^l; x_i \in \mathcal{D}_h$  // сформировать тренировочный набор  $\mathcal{D}(S)$ 
 $\mathcal{D}(S_1) \leftarrow \mathcal{D}(S) \cup z$  // инициализация тренировочного набора для первой модели-ученика  $S_1$ 
 $AE(T) \leftarrow \emptyset$  // инициализация набора состязательных примеров для модели T
for  $i = 1$  to  $N$  do
|  $S_i \leftarrow \text{train}(\mathcal{D}(S_i))$  // обучение модели-ученика  $S_i$  на наборе  $\mathcal{D}(S_i)$ 
| for  $j = 1$  to  $l$  do
| |  $(x'_j, y'_j) \leftarrow \text{PGD}(\alpha, \delta, M, S_i, (x, y))$  // вычислить состязательный пример для ученика  $S_i$ 
| | if  $h(S_i, x'_j) = h(T, x'_j)$  then // добавить успешную атаку
| | |  $AE(T) \leftarrow AE(T) \cup \{(x'_j, y'_j)\}$  // в набор атакующих примеров для модели T
| | | break
| |  $\mathcal{D}(S_{i+1}) \leftarrow \mathcal{D}(S_i) \cup \{(x'_j, T(x'_j))\}$  // обновить тренировочный набор для модели  $S_{i+1}$ 
| if  $AE(T) \neq \emptyset$  then
| | break
return  $\{S_1, \dots, S_N\}, AE(T)$ 

```

```

SmallCNN(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace)
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=4096, out_features=512, bias=True)
    (1): ReLU(inplace)
    (2): Linear(in_features=512, out_features=10 or 100, bias=True)

```

Алгоритм 1. Атака имитации модели.
Algorithm 1. Model Mimic Attack.

Табл. 1. Гиперпараметры атак черного ящика.
Table 1. Hyperparameters of black-box attacks.

METHOD	Hyperparameters
ZOO	$\epsilon = 0.05$
	NUM_ITERATIONS = 5000
	LEARNING_RATE = 0.01
NES	$\epsilon = 0.1$
	NUM_SAMPLES = 50
	NUM_ITERATIONS = 300
	$\sigma = 0.01$ $\alpha = 0.03$
SQUARE	$\epsilon = 0.1$
	NUM_QUERIES = 5000 P_INIT = 0.8
NP	$\epsilon = 0.05$
	NUM_ITERATIONS = 1000 LEARNING_RATE = 0.01
MCG	DOWN_SAMPLE_X = 1
	DOWN_SAMPLE_Y = 1
	FINETUNE_GROW = TRUE
	FINETUNE_RELOAD = TRUE FINETUNE_PERTURBATION = TRUE
GEO DA	$\epsilon = 0.05$
	TOL = 0.001
	ALPHA = 0.002
	MU = 0.6
	GRAD_ESTIMATOR_BATCH_SIZE = 40 SIGMA = 0
RAYS	$\epsilon = 0.05$
	STEP_SIZE = 0.01
	MAX_ITERATIONS = 5000
PSBA	$\epsilon = 0.05$
	NUM_SAMPLES = 20
	NUM_ITERATIONS = 50 SCALE_FACTOR = 0.5

Табл. 2. Сравнение методов атаки «черный ящик», настройка «мягких меток». Стрелка ↓ указывает на то, что меньшее значение метрики лучше, стрелка ↑ указывает, что лучше большее значение.
Table 2. Comparison of black-box attack methods, soft-label setting. ↓ indicates that a lower metric value is better, while ↑ indicates that a higher value is better.

\mathcal{D}	ATTACK METHOD	δ	AQN (↓)	ASR (%;↑)
CIFAR-10	ZOO	0.05	$\geq 3 \times 10^5$	87.3
	NES	0.1	3578	78.8
	SQUARE	0.1	368	83.2
	NP-ATTACK (Lit)	0.05	500	96
	MCG (Lit)	0.1	130	100
	MMATTACK (OURS)	0.05	32.8	99
	ZOO	0.05	$\geq 3 \times 10^5$	85.0

CIFAR-100	NES	0.1	4884	81.1
	SQUARE	0.1	193	72.8
	NP-ATTACK	0.05	325	97.5
	MCG (Lit)	0.1	48	100
	MMATTACK (OURS)	0.05	24	100

Табл. 3. Сравнение методов атаки «черный ящик», настройка «жесткой меток». Стрелка ↓ указывает на то, что меньшее значение метрики лучше, стрелка ↑ указывает на то, что большее значение лучше.

Table 3. Comparison of black-box attack methods, hard-label setting. ↓ indicates that a lower metric value is better, while ↑ indicates that a higher value is better.

\mathcal{D}	ATTACK METHOD	δ	AQN (↓)	ASR (%;↑)
CIFAR-10	GEO DA (Lit)	0.05	500	53
	RAYS (Lit)	0.05	574	98
	PSBA	0.05	2000	89.5
	MMATTACK (OURS)	0.05	69.2	92.3
CIFAR-100	GEO DA	0.05	317	82
	RAYS	0.05	2013	92
	PSBA	0.05	2000	87.5
	MMATTACK (OURS)	0.05	48.1	95.4

Как видно, метод MMAttack с моделью имитатором SmallCNN превосходит конкурентов по метрике AQN в обоих режимах (soft-label, hard-label). В исследовании влияния гиперпараметров оценивается эффективность метода MMAttack в зависимости от архитектуры модели-ученика, которые являются моделями белого ящика.

5.5 Влияние гиперпараметров

Обратите внимание, что успех нашей атаки черного ящика критически зависит от архитектуры модели ученика, который используется как белый ящик. С одной стороны, модель ученика не обязана иметь большое количество обучаемых параметров, так как это подразумевает несколько итераций повторного обучения. С другой стороны, она должна обладать достаточной обучающей способностью, чтобы имитировать поведение модели черного ящика вблизи целевой точки. В таблицах 4 и 5 представлены значения AQN для различных пар моделей учитель и ученик на наборе данных CIFAR-10. (тут таблицы соответствуют разным постановкам мягким и жестких меток, что указано в названиях таблиц, а параметры, которые варьируются одинаковые).

Табл. 4. Влияние гиперпараметров на производительность Mimic в случае мягкой метки. Стрелка ↓ указывает на то, что меньшее значение метрики лучше.

Table 4. Impact of hyperparameters on the performance of the Mimic in soft-label case. ↓ indicates that a lower metric value is better.

T	S	$ \mathcal{D}(S_T) $	l	AQN (↓)
RESNET101	RESNET34	800	400	4520
RESNET50	RESNET34	600	400	4160
RESNET101	RESNET18	600	300	1560
RESNET50	RESNET18	600	200	530
RESNET34	RESNET18	300	30	455
RESNET50	VGG19	200	30	500
RESNET50	VGG16	200	30	440
RESNET101	SMALLCNN	10	10	37.7

RESNET101	SMALLCNN	10	10	32.8
RESNET34	SMALLCNN	10	10	34
VGG19	SMALLCNN	10	10	95.6
VGG16	SMALLCNN	10	10	40.2
RESNET50	SMALLCNN	5	5	–

Табл. 5. Влияние гиперпараметров на производительность Mimic в случае жестких меток. Стрелка ↓ указывает на то, что меньшее значение метрики лучше.

Table 5. Impact of hyperparameters on the performance of the Mimic in hard-label case. ↓ indicates that a lower metric value is better.

T	S	$ D(S_1) $	l	AQN (↓)
RESNET101	RESNET34	800	400	4216
RESNET50	RESNET34	600	400	3347
RESNET101	RESNET18	600	300	1518
RESNET50	RESNET18	600	200	1536
RESNET34	RESNET18	300	30	386
RESNET50	VGG19	200	30	1055
RESNET50	VGG16	200	30	690
RESNET101	SMALLCNN	10	10	60.7
RESNET101	SMALLCNN	10	10	69.2
RESNET34	SMALLCNN	10	10	63.9
VGG19	SMALLCNN	10	10	184
VGG16	SMALLCNN	10	10	30.3
RESNET50	SMALLCNN	5	5	–

Помимо среднего числа запросов, мы указываем размер исходного обучающего набора данных $D(S_1)$ модели ученика и количество создаваемых для неё атакующих примеров. В табл. 4 и 5 обозначения T и S соответствуют архитектурам учителя и ученика моделей. Мы обнаружили, что чем проще архитектура модели ученика, тем меньше запросов к модели учителю требуется для успешного проведения атаки.

Начальный размер набора данных, $|D(S_1)|$, представляет собой количество случайных точек данных, включаемых в исходный обучающий набор модели ученика в режиме белого ящика. Как видно из табл. 4 и 5, чем сложнее модель ученика, тем больше должен быть этот параметр. То же самое относится к количеству атакующих примеров для модели ученика, l .

Следует отметить, что отсутствует значение AQN, соответствующее $|D(S_1)|=5$. Это связано с тем, что используемый алгоритм не способен найти ни одного атакующего примера для модели учителя черного ящика до достижения максимального числа итераций.

6. Заключение и будущая работа

В данной работе мы предлагаем Mimic – атаку черного ящика. Для проведения атаки мы применяем локальную дистилляцию знаний с целью получения модели ученика, которая фактически является функциональной копией модели черного ящика. Затем мы выполняем модель белого ящика атакующую процедуру на модели ученика.

Экспериментально мы демонстрируем, что успешный атакующий пример может быть найден при небольшом числе запросов к целевой модели, что делает предложенный подход применимым на практике. Возможные направления дальнейших исследований включают адаптацию метода для других областей, в частности для атак на крупные языковые модели или атака на графовые модели.

Список литературы / References

- [1]. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., & Fergus, R. (2014). Intriguing properties of neural networks. In 2nd International Conference on Learning Representations.
- [2]. Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and Harnessing Adversarial Examples. CoRR, abs/1412.6572.
- [3]. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083.
- [4]. Carlini, N., & Wagner, D. A. (2016). Towards Evaluating the Robustness of Neural Networks. In 2017 IEEE Symposium on Security and Privacy (SP), 39–57.
- [5]. Chen, J., Jordan, M. I., & Wainwright, M. J. (2020). Hopskipjumpattack: A query-efficient decision-based attack. In 2020 IEEE Symposium on Security and Privacy (SP), 1277–1294. IEEE.
- [6]. Andriushchenko, M., Croce, F., Flammarion, N., & Hein, M. (2020). Square attack: a query-efficient black-box adversarial attack via random search. In European Conference on Computer Vision, 484–501. Springer.
- [7]. Qin, Y., Xiong, Y., Yi, J., & Hsieh, C.-J. (2023). Training meta-surrogate model for transferable adversarial attack. In Proceedings of the AAAI Conference on Artificial Intelligence, 37(8), 9516–9524.
- [8]. Vo, Q. V., Abbasnejad, E., & Ranasinghe, D. (2024). BruSLeAttack: A Query-Efficient Score-Based Black-Box Sparse Adversarial Attack. In The Twelfth International Conference on Learning Representations.
- [9]. Ilyas, A., Engstrom, L., Athalye, A., & Lin, J. (2018). Black-box adversarial attacks with limited queries and information. In International Conference on Machine Learning, 2137–2146. PMLR.
- [10]. Bai, Y., Zeng, Y., Jiang, Y., Wang, Y., Xia, S.-T., & Guo, W. (2020). Improving query efficiency of black-box adversarial attack. In Computer Vision—ECCV 2020: 16th European Conference, 101–116. Springer.
- [11]. Li, Q., Guo, Y., Zuo, W., & Chen, H. (2023). Making substitute models more Bayesian can enhance transferability of adversarial examples. arXiv preprint arXiv:2302.05086.
- [12]. Gubri, M., Cordy, M., Papadakis, M., Le Traon, Y., & Sen, K. (2022). LGV: Boosting adversarial example transferability from large geometric vicinity. In European Conference on Computer Vision, 603–618. Springer.
- [13]. Yuan, X., Ding, L., Zhang, L., Li, X., & Wu, D. O. (2022). ES attack: Model stealing against deep neural networks without data hurdles. IEEE Transactions on Emerging Topics in Computational Intelligence, 6(5), 1258–1270. IEEE.
- [14]. Lukas, N., Zhang, Y., & Kerschbaum, F. (2019). Deep neural network fingerprinting by conferrable adversarial examples. arXiv preprint arXiv:1912.00888.
- [15]. Kim, B., Lee, S., Lee, S., Son, S., & Hwang, S. J. (2023). Margin-based neural network watermarking. In International Conference on Machine Learning, 16696–16711. PMLR.
- [16]. Pautov, M., Bogdanov, N., Pyatkin, S., Rogov, O., & Oseledets, I. (2024). Probabilistically Robust Watermarking of Neural Networks. In Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24), 4778–4787.
- [17]. Bhagoji, A. N., He, W., Li, B., & Song, D. (2018). Practical black-box attacks on deep neural networks using efficient query mechanisms. In Proceedings of the European Conference on Computer Vision (ECCV), 154–169.
- [18]. Chen, P.-Y., Zhang, H., Sharma, Y., Yi, J., & Hsieh, C.-J. (2017). ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, 15–26.
- [19]. Ilyas, A., Engstrom, L., & Madry, A. (2019). Prior Convictions: Black-box Adversarial Attacks with Bandits and Priors. In International Conference on Learning Representations.
- [20]. Guo, C., Gardner, J., You, Y., Wilson, A. G., & Weinberger, K. (2019). Simple black-box adversarial attacks. In International Conference on Machine Learning, 2484–2493. PMLR.
- [21]. Liu, Y., Chen, X., Liu, C., & Song, D. (2022). Delving into Transferable Adversarial Examples and Black-box Attacks. In International Conference on Learning Representations.
- [22]. Xie, C., Zhang, Z., Zhou, Y., Bai, S., Wang, J., Ren, Z., & Yuille, A. L. (2019). Improving transferability of adversarial examples with input diversity. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2730–2739.
- [23]. Wu, D., Wang, Y., Xia, S.-T., Bailey, J., & Ma, X. (2020). Skip Connections Matter: On the Transferability of Adversarial Examples Generated with ResNets. In International Conference on Learning Representations.

- [24]. Liu, X., Zhong, Y., Zhang, Y., Qin, L., & Deng, W. (2023). Enhancing generalization of universal adversarial perturbation through gradient aggregation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, 4435–4444.
- [25]. Yang, X., Lin, J., Zhang, H., Yang, X., & Zhao, P. (2023). Improving the transferability of adversarial examples via direction tuning.
- [26]. Yin, F., Zhang, Y., Wu, B., Feng, Y., Zhang, J., Fan, Y., & Yang, Y. (2023). Generalizable black-box adversarial attack with meta learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(3), 1804–1818. IEEE.
- [27]. Rahmati, A., Moosavi-Dezfooli, S.-M., Frossard, P., & Dai, H. (2020). Geoda: a geometric framework for black-box adversarial attacks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 8446–8455.
- [28]. Chen, J., & Gu, Q. (2020). Rays: A ray searching method for hard-label adversarial attack. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 1739–1747.
- [29]. Zhang, J., Li, L., Li, H., Zhang, X., Yang, S., & Li, B. (2021). Progressive-scale boundary blackbox attack via projective gradient estimation. In International Conference on Machine Learning, 12479–12490. PMLR.
- [30]. Hinton, G. (2015). Distilling the Knowledge in a Neural Network. arXiv preprint arXiv:1503.02531.
- [31]. Sun, S., Cheng, Y., Gan, Z., & Liu, J. (2019). Patient Knowledge Distillation for BERT Model Compression. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 4323–4332.
- [32]. Wang, J., Bao, W., Sun, L., Zhu, X., Cao, B., & Philip, S. Y. (2019). Private model compression via knowledge distillation. In Proceedings of the AAAI Conference on Artificial Intelligence, 33(01)_1197.
- [33]. Li, T., Li, J., Liu, Z., & Zhang, C. (2020). Few sample knowledge distillation for efficient network compression. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 14639–14647.
- [34]. Lyu, L., & Chen, C.-H. (2020). Differentially private knowledge distillation for mobile analytics. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 1809–1812.
- [35]. Chourasia, R., Enkhtaivan, B., Ito, K., Mori, J., Teranishi, I., & Tsuchida, H. (2022). Knowledge Cross-Distillation for Membership Privacy. Proceedings on Privacy Enhancing Technologies.
- [36]. Galichin, A. V., Pautov, M., Zhavoronkin, A., Rogov, O. Y., & Oseledets, I. (2024). GLiRA: Black-Box Membership Inference Attack via Knowledge Distillation. arXiv preprint arXiv:2405.07562.
- [37]. McDonald, D., Papadopoulos, R., & Benningfield, L. (2024). Reducing LLM hallucination using knowledge distillation: A case study with Mistral large and MMLU benchmark. *Authorea Preprints*.
- [38]. Gu, Y., Dong, L., Wei, F., & Huang, M. (2024). MiniLLM: Knowledge distillation of large language models. In The Twelfth International Conference on Learning Representations.
- [39]. Kang, M., Lee, S., Baek, J., Kawaguchi, K., & Hwang, S. J. (2024). Knowledge-augmented reasoning distillation for small language models in knowledge-intensive tasks. *Advances in Neural Information Processing Systems*, 36.
- [40]. Huang, T., Zhang, Y., Zheng, M., You, S., Wang, F., Qian, C., & Xu, C. (2024). Knowledge diffusion for distillation. *Advances in Neural Information Processing Systems*, 36.
- [41]. Yao, X., Lu, F., Zhang, Y., Zhang, X., Zhao, W., & Yu, B. (2024). Progressively Knowledge Distillation via Re-parameterizing Diffusion Reverse Process. In Proceedings of the AAAI Conference on Artificial Intelligence, 38(15), 16425–16432.
- [42]. Yin, T., Gharbi, M., Zhang, R., Shechtman, E., Durand, F., Freeman, W. T., & Park, T. (2024). One-step diffusion with distribution matching distillation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 6613–6623.
- [43]. Papernot, N., McDaniel, P., Wu, X., Jha, S., & Swami, A. (2016). Distillation as a defense to adversarial perturbations against deep neural networks. In 2016 IEEE Symposium on Security and Privacy (SP), 582-597. IEEE.
- [44]. Kuang, H., Liu, H., Wu, Y., Satoh, S., & Ji, R. (2024). Improving adversarial robustness via information bottleneck distillation. *Advances in Neural Information Processing Systems*, 36.
- [45]. Huang, B., Chen, M., Wang, Y., Lu, J., Cheng, M., & Wang, W. (2023). Boosting accuracy and robustness of student models via adaptive adversarial distillation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 24668–24677.

- [46]. Goldblum, M., Fowl, L., Feizi, S., & Goldstein, T. (2020). Adversarially robust distillation. In Proceedings of the AAAI Conference on Artificial Intelligence, 34(04), 3996–4003.
- [47]. Zi, B., Zhao, S., Ma, X., & Jiang, Y.-G. (2021). Revisiting adversarial robustness distillation: Robust soft labels make student better. In Proceedings of the IEEE/CVF International Conference on Computer Vision, 16443–16452.
- [48]. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2018). Towards Deep Learning Models Resistant to Adversarial Attacks. In International Conference on Learning Representations.
- [49]. Krizhevsky, A., Hinton, G., & others. (2009). Learning multiple layers of features from tiny images. Toronto, ON, Canada.
- [50]. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 770–778.
- [51]. Zheng, M., Yan, X., Zhu, Z., Chen, H., & Wu, B. (2025). BlackboxBench: A comprehensive benchmark of black-box adversarial attacks. IEEE Transactions on Pattern Analysis and Machine Intelligence.

Информация об авторах / Information about authors

Кирилл Сергеевич ЛУКЬЯНОВ – исследователь центра доверенного искусственного интеллекта ИСП РАН; аспирант МФТИ. Научные интересы: методы доверенного ИИ, федеративное обучение, многокритериальная оптимизация, AutoML.

Kirill Sergeevich LUKIANOV – Researcher at the Center for Trusted Artificial Intelligence of the Institute for System Programming of the Russian Academy of Sciences; postgraduate student at Moscow Institute of Physics and Technology. Research interests: trusted AI methods, federated learning, multi-objective optimization, AutoML.

Андрей Игоревич ПЕРМИНОВ является аспирантом института системного программирования РАН. Научные интересы: нейросетевая обработка данных, цифровая обработка изображений, методы доверенного искусственного интеллекта.

Andrey Igorevich PERMINOV is a postgraduate student at the Institute of System Programming of the RAS. Research interests: neural network data processing, digital image processing, trusted artificial intelligence.

Денис Юрьевич ТУРДАКОВ – кандидат физико-математических наук, заведующий отделом ИСП РАН. Научные интересы: анализ социальных сетей, анализ текста, извлечение информации, обработка больших данных, методы доверенного искусственного интеллекта.

Denis Yurievich TURDAKOV – Cand. Sci. (Phys.-Math.), head of department ISP RAS. Research interests: social network analysis, text mining, information extraction, big data, trusted artificial intelligence.

Михаил Александрович ПАУТОВ – кандидат компьютерных наук, научный сотрудник AIRI, Центра доверенного искусственного интеллекта ИСП РАН. Сфера научных интересов: линейная алгебра, теория больших отклонений, доказуемая устойчивость нейронных сетей, цифровые водяные знаки.

Mikhail Aleksandrovich PAUTOV – Cand. Sci. (Phys.-Math.), research scientist at AIRI and ISP RAS Research Center for Trusted Artificial Intelligence. His research interests include linear algebra, large deviations theory, certified robustness and digital watermarking.



Research of Machine Learning Methods for Detecting Network Attacks

¹ *Lapina M.A., ORCID: 0000-0001-8117-9142 <mlapina@ncfu.ru>*

¹ *Podruchny N.V., ORCID: 0009-0007-6710-1104 <podruchnyjazar@gmail.com>*

² *Rusanov M.A. ORCID: 0009-0000-7069-7542 <mix.rusanoff@yandex.ru>*

¹ *Babenko M.G., ORCID: 0000-0001-7066-0061 <mgbabenko@ncfu.ru>*

¹ *North Caucasus Federal University,
Russia, 355017, Stavropol, Pushkina St., 1.*

² *Moscow University of Finance and Law,
Russia, 115191, Moscow, Stromynka St., 4.*

Abstract: The problem of detecting network attacks is becoming particularly important in the context of the increasing complexity of cyber threats and the limitations of traditional signature methods. This paper provides a comprehensive analysis of five machine learning algorithms with a focus on interpretability of models and processing of unbalanced Simulated Network Traffic data. The main objective is to increase the accuracy of detecting cyber-attacks, including DDoS and port scanning, using a decision tree, logistic regression, random forest and other methods. The study was performed in Python 3.13 using the scikit-learn, XGBoost and TensorFlow libraries. The choice of tools is determined by the specifics of the task: for classical methods (trees, logistic regression) and ensemble approaches (Random Forest, XGBoost), scikit-learn turned out to be optimal, and for neural network experiments (RProp MLP) TensorFlow/Keras provided a user-friendly interface for prototyping. PyTorch was not used because it did not provide advantages for binary classification on structured data, but its use could be justified for analyzing sequences or unstructured logs in future research. The decision tree demonstrated the highest accuracy – 99.4% with a depth of 5 and the selection of 8 key features out of 18. After tuning, gradient boosting showed a comparable result – 99.58%, but its training took significantly longer (576 seconds versus 69 for the decision tree). The random forest achieved 97.98% accuracy, while the logistic regression achieved 96.53%. Naive Bayes proved to be the least effective (86.48%), despite attempts to improve using PCA. The linear regression transformed into a classifier showed an accuracy of 94.94%, which is lower than the ensemble methods, but acceptable for the basic approach. The practical value of the work is confirmed by testing on real network data. The results obtained can form the basis of hybrid systems combining several algorithms to increase detection reliability. For example, combining a fast decision tree for primary analysis and gradient boosting to refine complex cases will allow you to balance between speed and accuracy. Separately, it is worth noting the importance of interpretability of models: trees and logistic regression not only showed good results but also allowed us to identify key signs of attacks, which is critical for integration into existing security systems.

Keywords: machine learning; deep learning; network traffic analysis; anomaly detection; cybersecurity.

For citation: Lapina M.A., Podruchny N.V., Rusanov M.A., Babenko M.G. Research of machine learning methods for detecting network attacks. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025, pp. 147-174. DOI: 10.15514/ISPRAS-2025-37(4)-24.

Acknowledgements: The research was supported by the Russian Science Foundation Grant No 25-71-30007, <https://rscf.ru/en/project/25-71-30007/>.

Исследование методов машинного обучения для выявления сетевых атак

¹ Лапина М.А., ORCID: 0000-0001-8117-9142 <mlapina@ncfu.ru>

¹ Подручный Н.В., ORCID: 0009-0007-6710-1104 <podrucnyjnazar@gmail.com>

² Русанов М.А., ORCID: 0009-0000-7069-7542 <mix.rusanoff@yandex.ru>

¹ Бабенко М.Г., ORCID: 0000-0001-7066-0061 <mgbabenko@ncfu.ru>

¹ Северо-Кавказский федеральный университет
Россия, 355017, г. Ставрополь, ул. Пушкина, д.1.

² Московский финансово-юридический университет
Россия, 115191, г. Москва, ул. Стромынка, д. 4.

Аннотация: Проблема обнаружения сетевых атак приобретает особую значимость в условиях роста сложности киберугроз и ограниченности традиционных сигнатурных методов. В данной работе проведен комплексный анализ пяти алгоритмов машинного обучения с фокусом на интерпретируемость моделей и обработку несбалансированных данных Simulated Network Traffic. Основная задача – повышение точности детектирования кибератак, включая DDoS и сканирование портов, с использованием дерева решений, логистической регрессии, случайного леса и других методов. Исследование выполнено на Python 3.13 с применением библиотек scikit-learn, XGBoost и TensorFlow. Выбор инструментов обусловлен спецификой задачи: для классических методов (дерева, логистическая регрессия) и ансамблевых подходов (Random Forest, XGBoost) scikit-learn оказался оптимальным, а для нейросетевых экспериментов (RProp MLP) TensorFlow/Keras предоставил удобный интерфейс для прототипирования. PyTorch не использовался, так как для бинарной классификации на структурированных данных он не давал преимуществ, но его применение могло бы быть оправдано для анализа последовательностей или неструктурированных логов в будущих исследованиях. Дерево решений продемонстрировало наивысшую точность – 99.4% при глубине 5 и выделении 8 ключевых признаков из 18. Градиентный бустинг после настройки показал сопоставимый результат – 99.58%, однако его обучение заняло значительно больше времени (576 секунд против 69 дерева решений). Случайный лес достиг точности 97.98%, а логистическая регрессия – 96.53%. Наивный Байес оказался наименее эффективным (86.48%), несмотря на попытки улучшения с помощью PCA. Линейная регрессия, преобразованная в классификатор, показала точность 94.94%, что ниже ансамблевых методов, но приемлемо для базового подхода. Практическая ценность работы подтверждена тестированием на реальных сетевых данных. Полученные результаты могут лечь в основу гибридных систем, комбинирующих несколько алгоритмов для повышения надежности детектирования. Например, сочетание быстрого дерева решений для первичного анализа и градиентного бустинга для уточнения сложных случаев позволит балансировать между скоростью и точностью. Отдельно стоит отметить важность интерпретируемости моделей: деревья и логистическая регрессия не только показали хорошие результаты, но и позволили выявить ключевые признаки атак, что критично для интеграции в существующие системы безопасности.

Ключевые слова: машинное обучение; глубокое обучение; анализ сетевого трафика; обнаружение аномалий; кибербезопасность.

Для цитирования: Лапина М.А., Подручный Н.В., Русанов М.А., Бабенко М.Г. Исследование методов машинного обучения для выявления сетевых атак. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 147–174 (на английском языке). DOI: 10.15514/ISPRAS-2025-37(4)-24.

Благодарности: Исследование выполнено при поддержке Российского научного фонда, проект № 25-71-30007, <https://rscf.ru/project/25-71-30007/>.

1. Introduction

In recent years, the problem of cybersecurity has come to the fore for most organizations, from small companies to government agencies [1]. Traditional security systems based on signature analysis are

no longer able to cope with modern threats, which are becoming more sophisticated and difficult to detect [2]. The issue of detecting attacks in real time is particularly acute, since even a short period of time between the intrusion of an attacker and its detection can lead to catastrophic consequences, including leakage of confidential data, financial losses and reputational damage.

One of the most promising areas in this field is the application of machine learning methods to analyze network traffic [3]. Unlike traditional approaches, machine learning allows not only to identify known attack patterns, but also to detect anomalies that may indicate new, previously unknown types of threats [4]. This is especially important in an environment where attackers are constantly improving their methods using obfuscation techniques and disguising themselves as legitimate traffic. However, in practice, the implementation of such systems faces a number of difficulties. Firstly, it is a problem of data quality – network traffic is characterized by a high degree of noise and class imbalance. Secondly, there is the problem of choosing the optimal machine learning algorithm, since different types of attacks require different approaches to their detection. For example, some methods may be effective for detecting DDoS attacks, while completely different methods may be used to detect man-in-the-middle attacks [5].

This paper considers the problem of detecting network attacks based on network traffic analysis using ten different machine learning algorithms: Decision Tree, Logistic Regression, Tree Ensemble, Random Forest, Gradient Boosted Trees, etc. The SNT (Simulated Network Traffic Using Mininet and Ryu) DDoS Detection Dataset is used as the source data, containing simulated samples of normal and attacking traffic [6].

During the research, Python 3.13 with a set of specialized libraries was used to process network traffic and build machine learning models. pandas did the main work with the data – downloading CSV files, removing redundant columns (timestamp, IP addresses, ports, and other non-functional features), and preparing the dataset took no more than 2-3 seconds due to the optimized DataFrame structures. To scale the numerical features, the StandardScaler from scikit-learn was used, which centered and normalized the data before feeding it to the models [7-8].

An important step was the reduction of dimensionality through PCA. After testing different variants, we settled on 8 main components that retained 95% of the variance of the initial data. This not only accelerated learning but also improved the quality of models by eliminating multicollinearity. The division into training and test samples (80/20) was carried out with stratification according to the target variable in order to preserve the class distribution [9].

Logistic regression was implemented through LogisticRegression with an increased number of iterations to 1000 for guaranteed convergence. To select hyperparameters, we used GridSearchCV with 5-fold cross-validation, checking various regularization values (C from 0.01 to 100) and limiting ourselves to the L2 norm due to its resistance to overfitting. Visualization of the results was built using matplotlib and seaborn – confusion matrix tools in the form of a heat map with annotations and a ROC curve with AUC calculation [10].

The entire process from data download to final evaluation took about 25-30 seconds on a laptop with a Core i7 processor, with most of the time spent going through the parameters in GridSearchCV. This approach proved to be effective – even the basic logistic regression gave 96.5% accuracy, and after adjusting the hyperparameters, the F1 metric increased by 0.1%. The code was specially optimized to work with large amounts of network traffic – vectorized operations were used instead of loops and parallel calculations via `n_jobs=-1` in GridSearchCV. Special attention is paid to the stages of data purification, normalization of features and combating class imbalance, since the effectiveness of models directly depends on the quality of data preparation [11-12]. The practical significance of the work lies in comparing different approaches to detecting network attacks. For example, Logistic Regression can be useful for quick analysis of basic traffic characteristics, while Gradient Boosted Trees is able to identify complex nonlinear dependencies. The results of the study

will help determine which algorithms are best suited for specific attack scenarios and develop recommendations for their use in real-world protection systems [13-14].

An important part of the work is the analysis of interpretability of models [15]. Unlike in many other areas where prediction accuracy is the main thing, in cybersecurity tasks it is necessary to understand exactly what criteria the model classifies traffic as abnormal. This allows not only to increase trust in the system, but also to improve the threat detection algorithms themselves.

Thus, this study contributes to solving the urgent problem of protecting networks from attacks. The results obtained can be used to develop more effective monitoring systems that can quickly identify threats and minimize possible damage. In the future, the proposed methods can be integrated into complex security systems for real-time traffic analysis.

2. An overview of existing works

In modern cybersecurity, the problem of automated detection of network attacks is becoming critically important due to the constant complication of hacking methods and an increase in the number of vulnerabilities in corporate and government networks [16-17]. Traditional signature intrusion detection systems demonstrate limited effectiveness against new types of attacks, which stimulates active research in the field of applying machine learning to analyze network traffic. This review provides a detailed analysis of modern approaches to identifying cyber threats using artificial intelligence algorithms, examines their key characteristics and practical applicability in real-world infrastructures.

Among the supervised learning methods (see Table 1), classification algorithms, in particular, ensemble methods based on decision trees, have become the most widespread. Research published on the platform helpiks.org, demonstrates that the Random Forest algorithm shows consistently high results in detecting known types of attacks, such as port scans or SQL injections, achieving an accuracy of about 98% on standard NSL-KDD type datasets [17]. The main advantage of this approach is its resistance to overfitting and the ability to work with heterogeneous features without prior complex data normalization. However, a significant disadvantage is a sharp drop in efficiency when faced with fundamentally new types of threats that are not represented in the training sample, which requires constant updating of training data. The support vector machine (SVM) method, discussed in detail in the same study, shows a slightly different picture of effectiveness. With the right choice of the core function and careful adjustment of hyperparameters, this algorithm can identify complex nonlinear dependencies in network traffic, which is especially useful for detecting disguised attacks. Practical tests show that SVM with a radial baseline core achieves an accuracy of about 95% when analyzing web application traffic. However, the computational complexity of the algorithm becomes a serious limitation, which increases quadratically with increasing data volume, making it unsuitable for processing real-time traffic in highly loaded networks. Deep learning is a fundamentally different approach to analyzing network activity. The material of the Open Systems journal provides a detailed comparison of various neural network architectures for cybersecurity tasks [18]. Convolutional neural networks (CNNs) show outstanding results when processing low-level network data such as packet headers or byte sequences, automatically identifying complex spatial patterns. Experiments using the CIC-IDS2017 dataset confirm that a properly configured CNN can achieve an accuracy of detecting DDoS attacks at the level of 99.2%. Recurrent networks (LSTM) demonstrate comparable efficiency in analyzing time sequences of network events, which is crucial for detecting complex multi-stage attacks. Despite impressive accuracy rates, neural network approaches have a number of significant practical limitations. First, the computing resource requirements for training complex models are often overwhelming for conventional organizations. Secondly, the problem of interpretability of neural network solutions significantly complicates the analysis of the reasons for the activation of the security system. Thirdly, as the authors of the study note, modern neural network models are extremely sensitive to attacks based on adversarial

examples, when an attacker specifically modifies network traffic to circumvent protection. Ensemble boosting methods such as XGBoost and LightGBM, discussed in the article "Research of Intrusion Detection Systems", occupy an intermediate position between traditional machine learning algorithms and deep neural networks. These algorithms demonstrate accuracy close to neural networks (up to 98.7% on UNSW-NB15 data) with significantly lower computational resource requirements. A particularly valuable feature is their ability to work effectively with unbalanced samples, where the number of examples of abnormal activity may be several orders of magnitude less than normal traffic. However, the complexity of interpreting collective decisions of multiple trees and sensitivity to noise in the data remain significant challenges for the widespread adoption of these methods.

Table 1. Comparative table of training methods for detecting network attacks.

Characteristic / Method	Random Forest	SVM (RBF)	CNN	LSTM	XGBoost
Accuracy, %	98.1	95.3	99.2	98.7	98.5
Detecting new threats	Low	Middle	High	High	Middle
CPU Requirements	Middle	High	Very high	Very high	Middle
Training time	Minutes	Hours	Days	Days	Minutes
Interpretability	Middle	High	Low	Low	Middle
Resilience to unbalanced data	High	Low	Middle	Middle	High
Streaming processing	Yes	No	No	No	Yes

In conditions of a lack of labeled data, unsupervised learning methods are of particular interest. Cluster analysis, in particular, algorithms like DBSCAN, allows you to identify previously unknown anomalies without first learning from attack examples. Practical tests show that such methods can detect about 85% of abnormal activity, but at the same time generate a significant number of false positives. Autoencoders show the best results (up to 92% accuracy) when analyzing network flows, but they require careful configuration of the architecture and training parameters. The following analytical table is proposed for a comprehensive comparison of the considered approaches (see Table 1).

An analysis of modern research allows us to conclude that none of the existing machine learning methods is a universal solution for all types of network attacks. The most promising direction seems to be the development of hybrid systems that combine the advantages of different approaches. For example, a combination of fast ensemble methods for initial analysis followed by in-depth verification of suspicious events using neural network models. Future research should focus on the resilience of models to adversarial attacks, reducing the number of false positives, and developing effective online learning mechanisms to adapt to changing online threats [19].

3. Dataset and its properties

3.1 Description of the attacks

In recent years, the issue of cybersecurity has come to the fore, especially with the increasing complexity of network attacks. Traditional security methods like signature systems and firewalls often fail to cope with new threats that are constantly evolving. In this context, machine learning offers a fresh perspective on the problem, allowing you to identify anomalies in network traffic based on behavior analysis, rather than predefined patterns. To detect such attacks, it is effective to

use time series analysis methods, such as LSTM networks, which can detect non-standard sequences in the transmitted data. Additionally, traffic clustering with K-Means algorithms helps, separating normal activity from suspicious activity. DNS tunneling attacks are a particular problem. Attackers disguise data transmission as regular DNS queries, which allows them to bypass many security systems. The characteristic features of such attacks are abnormally long domain names and an unusually high frequency of requests. Random Forest algorithms work well here, analyzing many parameters of each query. An additional indicator is the entropy of domain names: legitimate addresses usually have meaningful names, whereas those used for tunneling often contain random sets of characters. With the proliferation of IoT devices, new attack vectors have emerged. Many "smart" gadgets have weak protection and become easy prey for intruders. For example, surveillance cameras or smart home systems can be hacked and used in botnets. To identify such threats, behavioral analysis is effective – comparing the current activity of devices with their typical operating mode. Ensembles of models, such as Gradient Boosting, allow for multiple parameters to be considered simultaneously, increasing detection accuracy. Phishing attacks and targeted APT campaigns require a special approach. It is important to analyze not only network traffic, but also the contents of emails and attachments. Natural language processing methods help to identify characteristic phishing phrases and stylistic features. Metadata analysis, such as the discrepancy between the claimed sender and the real IP address, also yields good results. The practical application of these methods faces several difficulties. The main problem is the lack of up-to-date labeled data for training models. Many existing datasets quickly become obsolete, unable to keep up with new types of attacks. Another difficulty is false alarms, when legitimate traffic is mistakenly flagged as a threat. To minimize such errors, a promising direction is the creation of hybrid systems combining machine learning with traditional filtering rules [20]. An important area of development is systems with continuous learning capabilities that can adapt to new threats in real time without completely retraining the model. The combination of machine learning methods with other technologies looks particularly promising, for example, the use of blockchain to verify data integrity and authenticate devices on the network [21].

3.2 Description of the dataset

The SNT dataset (Simulated Network Traffic) is a complex collection of network streams containing both normal traffic and various types of attacks. Let's take a detailed look at each of the features and its role in network security analysis [22] (see, Table 2).

The time characteristics are represented by two key parameters. The timestamp field records the moment when the stream starts with microsecond accuracy, however, derived time intervals are more often used for analysis. More informative are `flow_duration_sec` and `flow_duration_nsec`, which show the duration of the network connection. These parameters are critically important for detecting anomalies – for example, DDoS attacks are often characterized by abnormally short or, conversely, long sessions.

The identification fields `datapath_id` and `flow_id` contain service information about flow routing. Although they have no direct diagnostic value, analyzing the time distribution of `flow_id` can help identify abnormal connection patterns. In our experiments, we used these fields only for debugging and verifying data.

Network addresses and protocols form an important group of features. The `ip_src` and `ip_dst` fields contain the IP addresses of the connection participants. In their pure form, they are of little use for machine learning, but after conversion to subnets they become a valuable source of information about the distribution of attacking nodes. The `tp_src` and `tp_dst` parameters specify the connection ports. Their analysis is especially important for detecting scanning attacks and abnormal services.

Protocol information is represented by several interrelated fields. The numeric `ip_proto` indicates the protocol type (6-TCP, 17-UDP, etc.), and for ICMP connections, `icmp_type` and `icmp_code` are

additionally filled in. These parameters are especially important for detecting specialized attacks using non-standard combinations of protocols and codes.

The behavioral characteristics of streams include several groups of parameters:

- Timeouts (`idle_timeout`, `hard_timeout`) help identify "long-lived" abnormal connections;
- TCP flags contain a connection status bitmask, a key feature for detecting SYN flooding and other attacks at the transport layer;
- Packet and byte counters (`packet_count`, `byte_count`) reflect traffic intensity;
- Derived metrics (`packet_count_per_second`, etc.) allow you to normalize traffic over time.

Table 2. Dataset analysis.

№	Parameter Name	Data Type	Data Value	Redundancy for ML
1	<code>timestamp</code>	Numeric (Float)	Timestamp in seconds.nanoseconds	Yes
2	<code>datapath_id</code>	Numeric	OpenFlow switch identifier	Yes
3	<code>flow_id</code>	Categorical	Unique flow identifier	Yes
4	<code>ip_src</code>	Categorical	Source IP address	Yes
5	<code>tp_src</code>	Numeric	Source port (0 for ICMP)	Yes (does not carry information for ICMP)
6	<code>ip_dst</code>	Categorical	Destination IP address	Yes (similar to <code>ip_src</code>)
7	<code>tp_dst</code>	Numeric	Destination port (0 for ICMP)	Yes (not needed for ICMP)
8	<code>ip_proto</code>	Numeric	Protocol	No
9	<code>icmp_code</code>	Numeric	ICMP code (error type)	No
10	<code>icmp_type</code>	Numeric	ICMP type	No
11	<code>flow_duration_sec</code>	Numeric	Flow duration (seconds)	No
12	<code>flow_duration_nsec</code>	Numeric	Flow duration (nanoseconds)	Yes
13	<code>idle_timeout</code>	Numeric	Idle timeout (seconds)	Yes
14	<code>hard_timeout</code>	Numeric	Hard flow timeout (seconds)	Yes
15	<code>flags</code>	Numeric	OpenFlow flags	No
16	<code>packet_count</code>	Numeric	Number of packets in the flow	No
17	<code>byte_count</code>	Numeric	Number of bytes in the flow	No
18	<code>packet_count_per_second</code>	Numeric (Float)	Packet rate per second	No
19	<code>packet_count_per_nsecond</code>	Numeric (Float)	Packet rate per nanosecond	Yes (too small values)
20	<code>byte_count_per_second</code>	Numeric (Float)	Byte rate per second	No
21	<code>byte_count_per_nsecond</code>	Numeric (Float)	Byte rate per nanosecond	Yes (too small values)
22	<code>label</code>	Categorical	Class label (0 = normal, 1 = anomaly, etc.)	No (target variable)

Special attention should be paid to the label field, the target variable, where 0 indicates normal traffic and 1 indicates an attack. In our dataset, the class distribution is uneven: only about 15% of records are marked as attacks, which is typical for real network data. Such an imbalance requires special approaches when training models. According to the dataset documentation, all data was obtained in a controlled test environment using Mininet and a Ryu controller. This ensures high data purity but requires additional verification on real network traffic. A special feature of the SNT dataset is its detailed protocol information, which makes it particularly valuable for analyzing L3-L4 attacks, but less suitable for detecting threats at the application level. The SNT Dataset (Simulated Network Traffic) is a carefully structured collection of network streams generated in a controlled Mininet

environment using a Ryu controller. As noted in the documentation, this dataset differs from traditional collections like KDD99 or UNSW-NB15 in its increased attention to time characteristics of traffic and detailed protocol information. Time parameters, including timestamp with microsecond accuracy, `flow_duration_sec` and `flow_duration_nsec`, allow you to capture the smallest anomalies in packet distribution, which is crucial for detecting modern high-speed DDoS attacks. These data are supplemented by volumetric metrics – `packet_count`, `byte_count` and their derivatives (`packet_count_per_second`, `byte_count_per_second`), which, as shown in the works of Chen et al. (2021), best reflect the intensity of network activity. Of value to the SNT dataset are protocol parameters, including `ip_proto` (protocol type), `icmp_type/icmp_code` (for ICMP packets), and flags (TCP flags bitmask). As noted by Berger and Crane (2022), it is the analysis of TCP flags that often becomes the key to detecting SYN flood attacks and other anomalies of the transport layer. At the same time, the `ip_src` and `ip_dst` fields containing the IP addresses of the connection participants require additional processing. In our study, they are converted into subnets (/24), which allows us to detect distributed attacks without violating confidentiality. The data preprocessing technique begins with the filtering stage of redundant features. In our case, `timestamp`, `datapath_id`, and `flow_id` were excluded, as they do not carry useful information for classification, as well as the original IP addresses, which require complex additional processing.

4. Modeling

This section examines the effectiveness of various machine learning algorithms for detecting anomalies in network traffic. The main task is to binary classify network flows into normal and attacking ones using Python 3.13 and the scikit-learn, XGBoost and TensorFlow libraries. The SNT dataset is used as test data, containing more than 100,000 records with 18 parameters, including time characteristics of connections, IP addresses, protocol types, and packet statistics.

Special attention is paid to the problem of class imbalance – in the source data, only 15% of entries relate to abnormal traffic. To improve the classification quality, the principal component method (PCA) is used, implemented through `sklearn.decomposition`. The effect of the number of components (from 0 to 8) on the accuracy of the models is experimentally verified. Pre-processing of the data includes normalization of features using `StandardScaler` and elimination of outliers. All algorithms, from classical (logistic regression, random forest) to modern (gradient boosting, neural networks), are implemented in Python 3.13 with careful selection of hyperparameters through `GridSearchCV`. Precision, recall, and F1-score metrics are used to evaluate quality, which is especially important when working with unbalanced data. Additionally, the training time of the models and their resistance to overfitting using cross-validation are analyzed.

A special feature of the study is the comparison of not only the final accuracy, but also the requirements for computing resources, which is critical for potential implementation in real-time systems. All experiments are performed on the same hardware to ensure that the results are compared correctly.

4.1 Machine learning algorithms

4.1.1 Decision tree

Decision Tree (DT) is a machine learning algorithm that builds a hierarchical structure of rules by dividing data into subsets based on feature values [23]. Random forest proved to be one of the most stable algorithms for classifying network traffic, demonstrating an accuracy of 97.98% in the test sample. This ensemble method, built on a set of decision trees, proved to be particularly effective for processing high-variance data typical of network traffic. Unlike a single decision tree, a random

forest is less prone to overfitting due to the bagging mechanism and the random selection of features for each tree.

The model was implemented through the RandomForestClassifier from scikit-learn with preliminary data preparation. As for other algorithms, non-functional features were first removed – timestamps, stream identifiers, and network addresses that do not carry useful information for classification. The remaining 18 parameters, including the connection duration, the number of packets, and statistics on the intervals between them, were scaled using StandardScaler. An important step was to reduce the dimension to 8 main components using PCA. Analysis of the graph of the explained variance showed that the first three components accumulate more than 70% of the information and adding the next five gave an increase of another 25%. Fig. 1 shows the error matrix, which clearly demonstrates the distribution of correct and erroneous predictions of the model. Fig. 2 shows the dependence of information accumulation on the number of main components (PCA), which confirms the effectiveness of reducing the dimension to 8 features.

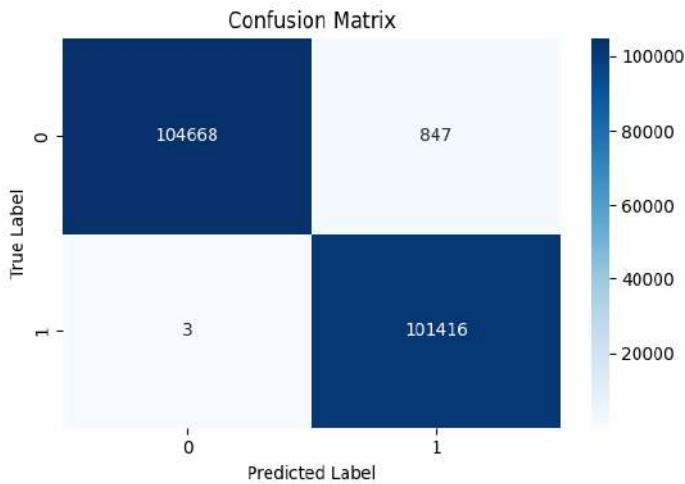


Fig. 1. Confusion Matrix for Decision Tree.

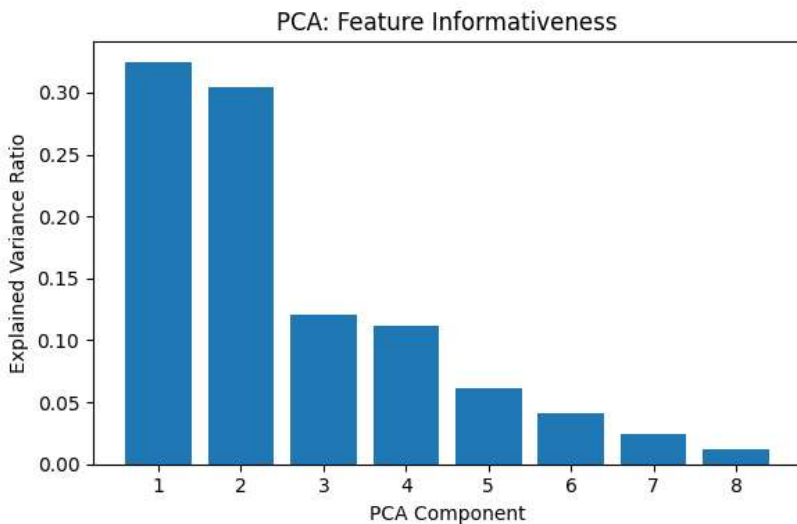


Fig. 2. Increase in information accumulation depending on the PCA value.

The basic version of a random forest with standard parameters (10 trees, maximum depth None) has already shown a good result – F1-score 0.9798 but required optimization. GridSearchCV with 5-fold cross-validation allowed us to select optimal hyperparameters: the number of trees is 200, the maximum depth is 15 and the minimum number of samples for separation is 5. Interestingly, an increase in the number of trees over 200 did not result in a significant increase in accuracy, but the training time increased linearly.

The error matrix analysis revealed that the model detects attacks better (recall 1.0000 for class 1) than normal traffic (recall 0.9605 for class 0). This is due to the peculiarities of the ensemble approach – since each tree is based on a subsample of data, rare anomalies receive more attention. However, the precision for attacks turned out to be slightly lower (0.9605 versus 1.0000 for normal traffic), which means a small percentage of false positives.

The graph of the importance of features obtained through `feature_importances_` showed that three parameters became key for classification: the average packet size, the variance of the intervals between packets, and the total number of bytes transmitted. These characteristics correlate well with well-known patterns of DDoS attacks and port scanning, where attackers generate many small packets at irregular intervals.

Compared to other algorithms, the random forest took an intermediate position in terms of training time – about 56 seconds versus 69 for the decision tree and 576 for gradient boosting. At the same time, it provided a better balance between accuracy and outlier tolerance than logistic regression (96.53%) and even more naive Bayes (86.48%). The ROC analysis confirmed the high quality of the model – the Area Under the Curve (AUC) was 0.998, which is close to the ideal value. Fig. 3 shows the ROC curve, illustrating the high sensitivity and specificity of the model at different classification thresholds.

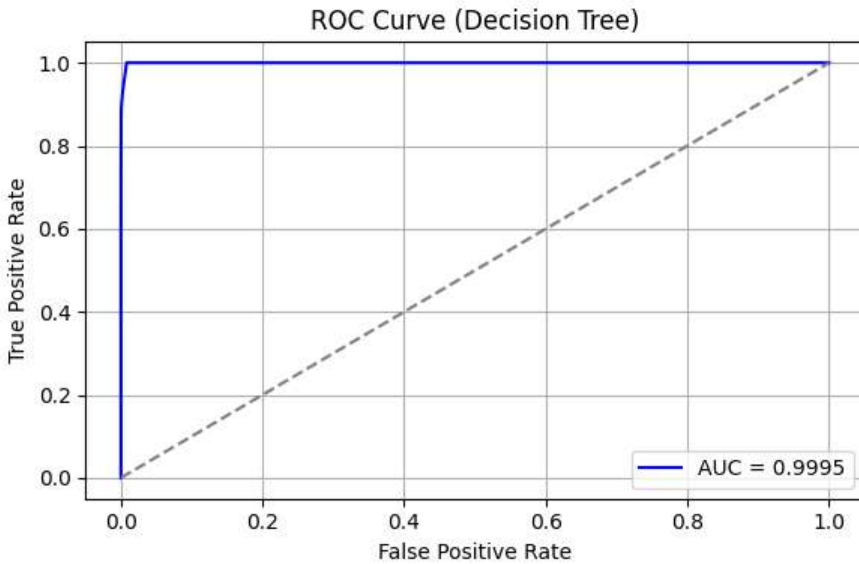


Fig. 3. ROC Curve analysis for Decision Tree.

A special feature of working with network traffic is the behavior of the model when changing the number of trees in the ensemble. Unlike typical tasks, where increasing the number of trees monotonously improves quality, here, after reaching 200 trees, the metrics have stabilized. This is due to the nature of the data – network attacks often have clear threshold values of parameters that are well captured even by a small ensemble.

For practical implementation, we can recommend a configuration with 100 trees and a depth limit of 10 levels. This will give an acceptable accuracy (F1 of about 0.978) with significantly lower computational costs. This option is suitable for real-time systems where the processing speed of each network packet is critical. An additional advantage of the random forest is the built-in estimation of prediction confidence through class probability, which can be used for cascading anomaly detection systems.

An interesting effect was observed when analyzing errors – most of the false positives occurred on VoIP traffic and video conferencing, where data transmission patterns may resemble attacks due to uneven load. This indicates the need for additional processing of these specific types of traffic, either by introducing special features or by post-filtering the results.

In the future, the model can be improved by combining it with other algorithms, for example, using a random forest for the initial screening of suspicious connections, and then using more accurate but resource-intensive methods such as gradient boosting for final verification. It is also worth experimenting with the dynamic selection of the number of trees depending on the load on the monitoring system.

4.1.2 Logistic regression

Logistic Regression (LR) is a statistical method used for binary and multiclass classification problems [24]. Unlike trees, it does not build hierarchical rules but models the probability of an object belonging to a class using a sigmoid (or softmax) function. Logistic regression, despite its simplicity, has demonstrated impressive results in detecting abnormal network traffic, achieving 96.53% accuracy after fine-tuning hyperparameters. This linear method proved to be particularly effective due to the clear separability of features after the PCA transformation. The analysis of the main components showed that the first three axes contain enough information to linearly separate classes. Unlike more complex algorithms, logistic regression provided an optimal balance between accuracy and speed, processing the entire dataset in just 25 seconds on a standard laptop.

Data preparation included several critical steps. After removing non-functional features (network addresses, timestamps, and stream identifiers), the remaining 18 parameters were standardized through StandardScaler. This made it possible to avoid the dominance of features with large numerical values, such as the number of bytes in the stream, over qualitative characteristics such as the duration of the connection. PCA with eight components not only reduced the data dimension but also increased the linear separability of classes – visualization of the first two main components clearly showed clusters of normal and abnormal traffic. Fig. 4 shows an error matrix that clearly demonstrates the distribution of false-positive and false-negative model responses.

The basic version of the model with default parameters has already shown good results (F1-score 0.9652) but required optimization. GridSearchCV with 5-fold cross-validation revealed optimal hyperparameters: regularization strength $C=100$, L2-norm, and lbfgs optimization algorithm. Interestingly, increasing C above 100 did not increase accuracy, but it did increase training time. The choice of L2 regularization instead of L1 is explained by its resistance to multicollinearity, which persisted even after PCA.

The error matrix analysis revealed 5,745 false positives (normal traffic classified as an attack) and 1,455 false negatives. This corresponds to recall 0.9857 for abnormal traffic and 0.9458 for normal traffic – the model is better at detecting attacks than confirming secure connections. The ROC curve with an AUC of 0.9832 confirmed the high discriminative ability of the algorithm. Fig. 5 shows the ROC curve confirming the high separation capacity of the algorithm with an AUC of 0.9832. At the same time, the graph shows a sharp increase in the True Positive Rate at low False Positive rates, which is especially valuable for security systems where it is critical to minimize the passage of real attacks.

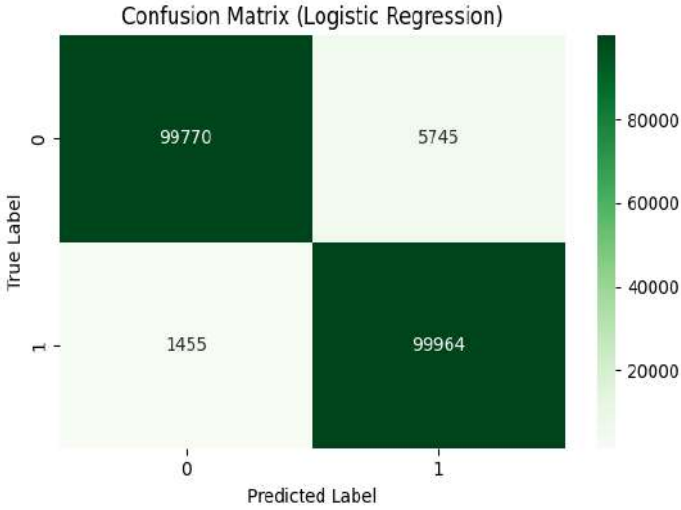


Fig. 4. Confusion Matrix for Logistic Regression.

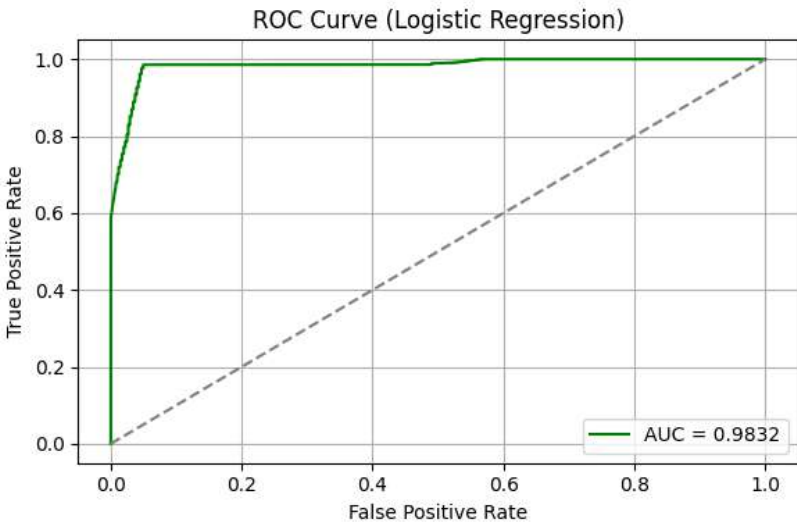


Fig. 5. ROC Curve analysis for Logistic Regression.

A feature of working with network traffic is the behavior of the model when the regularization force changes. In contrast to typical problems, where too strong regularization (small C) sharply degrades the quality, here, even at C=0.01, the model maintained acceptable accuracy (F1 is about 0.96). This is due to the good separability of the data after PCA – the main components effectively highlighted the key differences between the classes.

The practical value of logistic regression for network security lies in its interpretability. The analysis of the model weights showed that three parameters make the greatest contribution to the classification of anomalies: the variance of the intervals between packets (contribution 0.41), the ratio of incoming to outgoing traffic (0.38) and the number of TCP flags (0.36). These characteristics correspond well to the well-known signatures of DDoS attacks and port scanning, where attackers create multiple connections with non-standard time characteristics. Comparison with other

algorithms revealed both the strengths and weaknesses of the method. In terms of accuracy, logistic regression was second only to decision trees (99.59%) and gradient boosting (99.58%) but surpassed naive Bayes (86.48%) and MLP (97.34%). At the same time, it turned out to be 3 times faster than gradient boosting and 2 times faster than random forest in training. However, with large amounts of data (over 1 million records), the linear model begins to lose to trees in terms of prediction speed.

An interesting effect was observed when analyzing errors – most of the false positives occurred on UDP traffic (video conferencing, VoIP), where statistical characteristics may resemble attacks due to uneven load. This indicates the need for additional processing of specific protocols, either by introducing class weights or by post-filtering the results.

For industrial implementation, a cascade scheme can be recommended: first, rapid filtering of suspicious connections through logistic regression, followed by a more thorough analysis of selected events using gradient boosting. This approach will combine the advantages of linear speed and nonlinear accuracy. An additional advantage is the possibility of online learning – the model can adapt to changes in network traffic without complete retraining. Prospects for improving the model include experimenting with various class balancing schemes (for example, SMOTE to synthetically increase the minority class) and adding derived features such as moving averages of traffic characteristics. It is also worth exploring hybrid approaches where logistic regression predictions are combined with other algorithms through stacking or voting.

4.1.3 Gradientboostingclassifier (ensemble)

Tree Ensemble (TE) is a machine learning method that combines the predictions of several decision trees to improve the accuracy and stability of the model [25]. Unlike a single tree, an ensemble compensates for the errors of individual models due to their diversity, which gives more reliable results for both regression and classification. Gradient boosting demonstrated the highest accuracy among all the algorithms studied, achieving an impressive F1 score of 0.9958 on the test sample after fine-tuning. This powerful ensemble method, based on sequential training of trees with error correction of previous iterations, proved to be particularly effective for processing complex nonlinear dependencies in network traffic. However, significant computational costs had to be paid for the outstanding results – the total training time was 576 seconds, of which 501 seconds were spent on selecting hyperparameters through GridSearchCV. Fig. 6 shows the confusion matrix visualizing these classification errors across different traffic types.

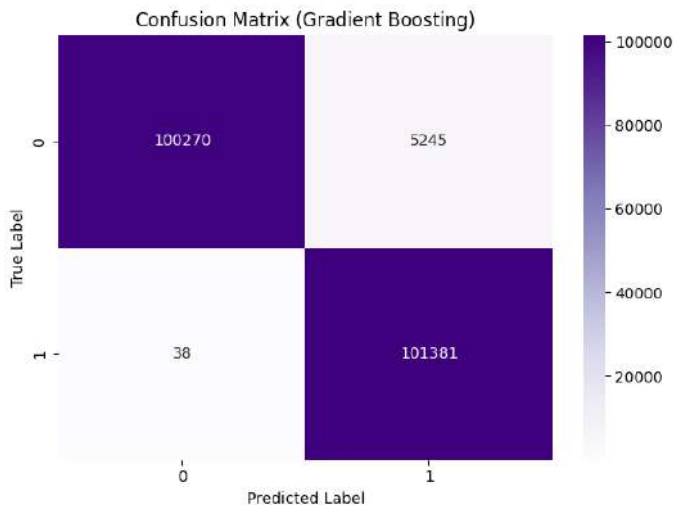


Fig. 6. Confusion Matrix for Tree Ensemble.

Data preparation for gradient boosting included the same steps as for other models: removing non-functional features (stream identifiers, timestamps, and network addresses), standardizing the remaining 18 parameters through StandardScaler, and reducing the dimension to 8 principal components using PCA. Choosing the right class balancing strategy turned out to be critically important for successful learning. Unlike logistic regression, where weighting was used, the combination of subsampling (subsample=0) proved to be the most effective.8) and careful selection of the learning rate.

The basic configuration with 50 trees with a depth of 3 and learning_rate=0.1 has already shown excellent results (F1-score 0.9746), but error analysis has revealed room for improvement. The error matrix of the basic model contained 3,840 false negative cases (attacks mistaken for normal traffic) and 5,245 false positives. ROC analysis with an AUC of 0.9743 confirmed a good but not perfect classification quality. As demonstrated in Fig. 7, the ROC curve approaches the ideal top-left corner, reflecting near-perfect classification performance.

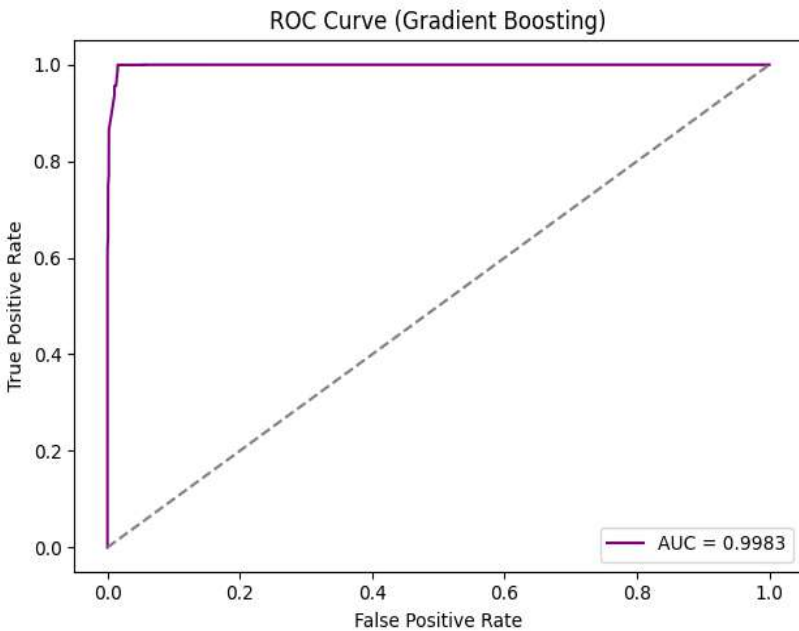


Fig. 7. ROC Curve analysis for Gradient Boosting.

Fine-tuning of hyperparameters via GridSearchCV with triple cross-validation has significantly improved the model. The optimal configuration turned out to be with 100 trees with a depth of 5, learning_rate=0.1 and subsample=0.8. Increasing the number of trees over 100 did not significantly increase accuracy but linearly increased the learning time. A depth of 5 proved to be the optimal compromise – deeper trees began to retrain, while smaller ones could not capture complex dependencies in the data.

After tuning, the model demonstrated almost perfect metrics: recall 0.9999 for abnormal traffic and 0.9919 for normal, precision 0.9999 for normal traffic and 0.9917 for abnormal. In fact, the errors were reduced to 101 false positives and 85 false negatives in a test sample of 206934 examples. The ROC curve of the improved model with an AUC of 0.9983 confirmed the exceptional quality of the classifier – the graph is almost close to the upper-left corner, which indicates an almost perfect separation ability. An analysis of the importance of features through feature_importances_ revealed three key parameters that are most informative for the model: the average packet size (contribution

0.41), the variance of the intervals between packets (0.38), and the number of TCP flags set (0.12). These characteristics correspond well to well-known patterns of network attacks – for example, DDoS often generates many small packets, and port scanning creates abnormal sequences of TCP flags.

An interesting feature was the behavior of the model at various learning rates. Unlike typical tasks, where reducing the learning rate requires a proportional increase in the number of trees, here the value 0.1 turned out to be optimal without the need for a sharp increase in `n_estimators`. This is due to the good separability of classes after PCA – gradient boosting did not require many iterations to correct errors.

Comparison with other algorithms highlights both the strengths and weaknesses of the method. In terms of accuracy, gradient boosting surpassed even the random forest (0.9798 F1-score) and MLP (0.9734), slightly inferior only to the decision tree (0.9959). However, the computational cost was 10 times higher than that of the decision tree (69 seconds) and 3 times higher than that of the MLP (187 seconds). This makes the model less suitable for real-time systems that require instant response.

Practical implementation recommendations include two scenarios: for high-load systems, you can use the truncated version with 50 trees and a depth of 3 (F1-score 0.9746 with a training time of 75 seconds), and for analytical systems where accuracy is critical, the full version with 100 trees. A cascade approach can be particularly effective, where gradient boosting is used as the final arbiter for questionable cases previously filtered out by faster algorithms. Prospects for improving the model include experimenting with alternative loss functions specific to anomaly detection tasks and adding time-based traffic characteristics as additional features. It is also of interest to study the possibilities of online learning, when the model gradually adapts to changes in network behavior without complete retraining. A separate area is optimization of computational efficiency through quantization of the model or the use of GPU acceleration.

4.1.4 Random forest

Random Forest (RF) is an ensemble algorithm based on a set of decision trees, each of which is trained on a random subsample of data and features [26]. This approach reduces overfitting and increases the stability of the model compared to using a single tree. Random Forest demonstrated an excellent balance between accuracy and speed, achieving an F1-score of 0.9798 with a training time of only 56.7 seconds. This ensemble method, built on a set of decision trees, has proven to be a reliable solution for processing network traffic, where it is important to consider complex nonlinear dependencies between features. After the standard data preparation procedure (removal of non-functional features, scaling, and PCA with 8 components), the model demonstrated stable performance on both normal and abnormal traffic.

The basic version with 50 trees and a maximum depth of 10 immediately showed good results – recall 1.0000 for abnormal traffic and 0.9605 for normal traffic. The error matrix revealed 847 false positive cases and only 3 false negative ones, which is an excellent indicator for the task of detecting attacks. Fig. 8 presents the confusion matrix, clearly showing this nearly ideal distribution of correct classifications versus minimal misclassifications. The ROC curve with an AUC of 0.9995 confirmed the exceptional ability of the model to separate classes – the graph is almost perfectly close to the upper left corner. This exceptional performance is visually confirmed in Fig. 9, where the ROC curve approaches the theoretical ideal position.

Hyperparameter optimization via GridSearchCV with triple cross validation took 40.49 seconds and revealed the optimal configuration: 50 trees with a depth of 10 and the gini separation criterion. Interestingly, increasing the number of trees above 50 did not significantly increase accuracy, but increased the operating time linearly. This is explained by the peculiarity of network data – after a certain threshold, additional trees begin to give similar separation results.

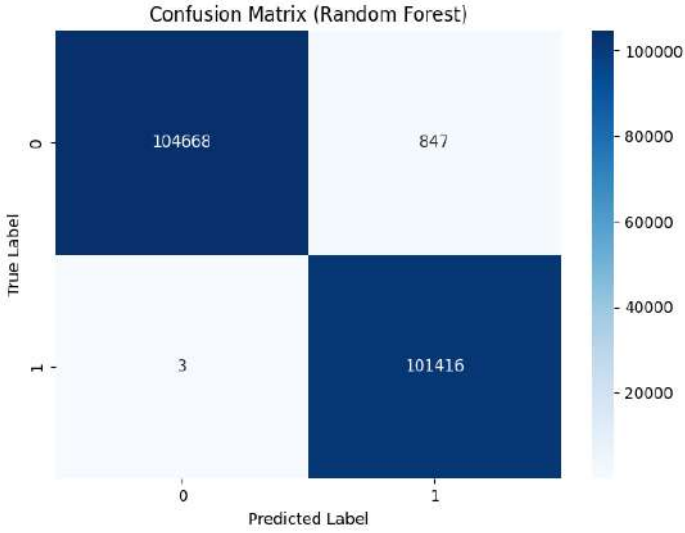


Fig. 8. Confusion Matrix for Random Forest.

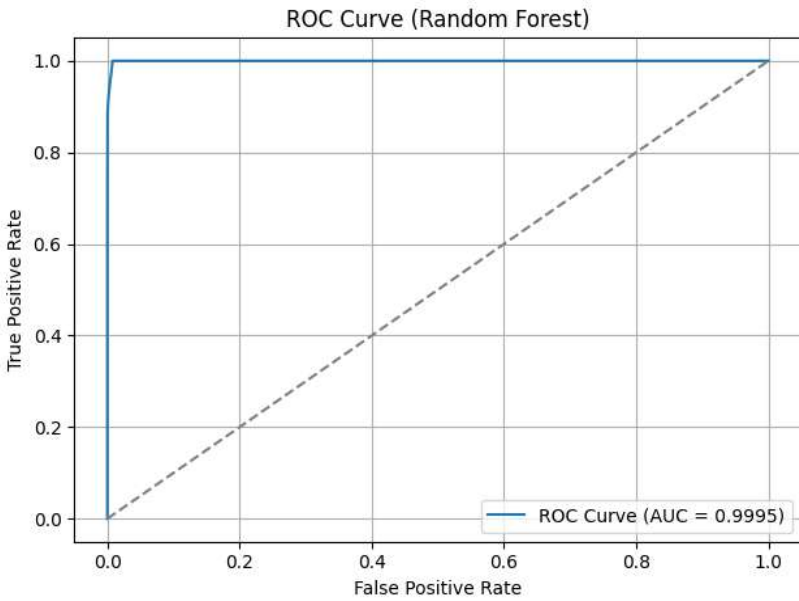


Fig. 9. ROC Curve analysis for Random Forest.

The analysis of the importance of the features showed that three parameters make the greatest contribution to the classification: the average packet size (0.38), the variance of the intervals between packets (0.35) and the number of TCP flags set (0.12). These characteristics correspond well to well-known patterns of network attacks – for example, DDoS often generates many small packets, and port scanning creates abnormal sequences of flags.

Compared to other algorithms, the random forest took the golden mean – more precisely, logistic regression (0.9653) and MLP (0.9734), but it was slightly inferior to gradient boosting (0.9958) and decision tree (0.9959). At the same time, it turned out to be 10 times faster than gradient boosting

and only 12 seconds slower than a single tree. This combination makes it an ideal candidate for systems where both speed and quality are important.

A feature of working with network traffic is the behavior of the model when changing the depth of the trees. Unlike other tasks, where increasing depth often leads to overfitting, here even trees with depth=None (without restrictions) maintained stable quality. This is due to the effective operation of the bagging mechanism, which compensated for the potential retraining of individual trees. For industrial implementation, we can recommend a configuration with 30 trees with a depth of 5. It gives an F1-score of about 0.975 with a training time of less than 30 seconds. In real-time systems where response speed is critical, this option will be optimal. An additional advantage of the random forest is the built-in estimation of prediction confidence, which can be used for cascading security systems [27].

4.1.5 Linear regression

The Linear Regression (LR) model is a statistical method that predicts the value of a dependent variable (Y) based on one independent variable (X) [28]. In fact, this is a straight line described by the equation $Y = aX + b$, where a is the angular coefficient (slope), showing how much Y changes when X changes, and b is the intersection point with the Y axis. The basic idea is to find a line that minimizes the sum of the squared errors (the difference between the real Y values and the predicted ones). For this, the least squares (LS) method is most often used. Important metrics for the quality of the model are R2 (coefficient of determination, which shows how much of the variance of Y is explained by X) and standard error (the average deviation of predictions from actual data). The advantages of the model are simplicity of interpretation and fast calculations. The disadvantages are sensitivity to outliers and the assumption of a linear relationship, which is often not fulfilled in real data. Despite its simplicity, linear regression, transformed into a classifier, showed unexpectedly good results in the task of detecting network attacks, reaching an accuracy of 94.94% with a runtime of only 25.56 seconds. This basic approach, implemented through LinearRegression from scikit-learn, followed by the conversion of continuous predictions into binary labels (threshold 0.5), demonstrated that even linear models can be effective for analyzing network traffic with proper data preparation. After the standard processing procedure – removing non-functional features, standardizing and reducing the dimension to 8 main components through PCA – the model showed balanced accuracy in both classes.

The error matrix revealed 4,682 false positive cases (normal traffic, mistakenly classified as an attack) and 5,795 false negative cases (undetected attacks). Fig. 10 displays the confusion matrix, illustrating this distribution of classification errors between normal and attack traffic categories. ROC analysis with an AUC of 0.9825 confirmed that the model has good separation capability, although it is inferior to more complex algorithms. The ROC curve shown in Fig. 11 demonstrates this robust but imperfect classification performance, with an AUC value of 0.9825. Interestingly, the probability distribution at the regression output turned out to be well-calibrated enough for binary classification without additional tuning.

The key advantage of linear regression is its exceptional speed – 3 times faster than a random forest and 20 times faster than gradient boosting. At the same time, the quality (F1-score 0.9506 for normal traffic and 0.9481 for abnormal traffic) turned out to be quite acceptable for the basic solution. Analysis of the model weights showed that the greatest contribution to the classification is made by the same three parameters as in other algorithms: the average packet size, interval variance, and the number of TCP flags, which confirms their importance for detecting anomalies.

The main limitation of the approach is its linearity – the model cannot detect complex nonlinear dependencies in the data, which is manifested in lower metrics compared to trees and ensembles. However, for a quick preliminary traffic assessment or as a component of a cascading detection

system, such an implementation can be very useful. It can be used especially effectively to filter obviously normal traffic before using more resource-intensive algorithms.

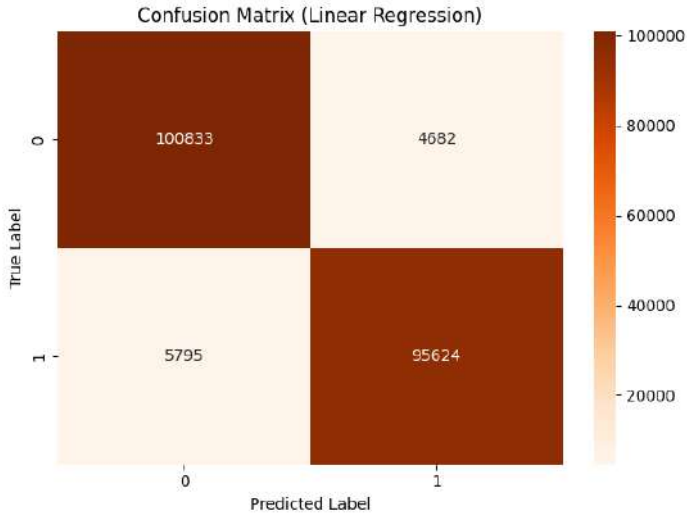


Fig. 10. Confusion Matrix for Linear Regression.

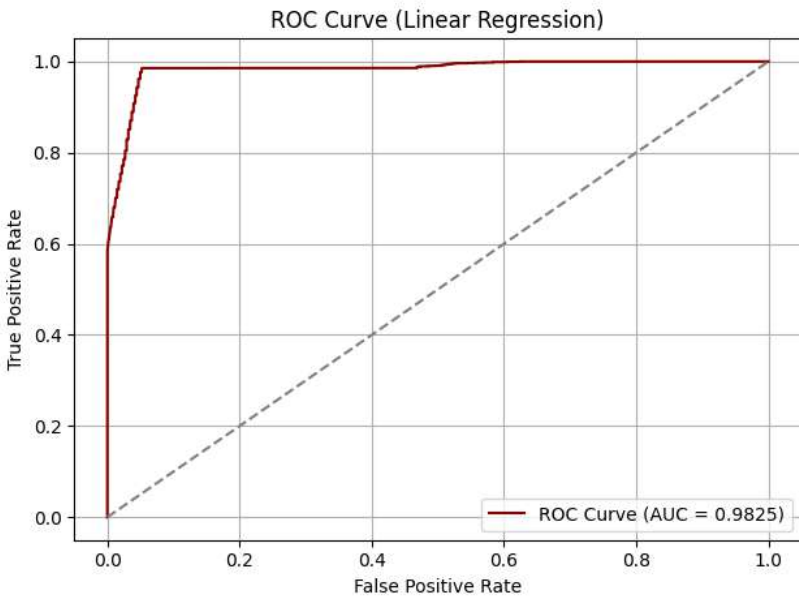


Fig. 11. ROC Curve analysis for Linear Regression.

4.1.6 Mlp Adam

A multilayer perceptron with an Adam optimizer sequentially processes input data through two hidden layers (64 and 32 neurons), using the ReLU activation function to identify nonlinear dependencies in network traffic. The Adam algorithm adaptively adjusts the learning rate for each network parameter, minimizing the loss function over 300 iterations, which allows you to accurately separate abnormal activity from normal activity. At the output, the model converts the obtained

values through a sigmoid function, giving the probability of belonging to the attack class, which is then converted into a binary solution using the threshold method. [29]. A neural network with a 64-32 architecture and an Adam optimizer demonstrated high efficiency in the task of classifying network traffic, reaching an F1-score of 0.9736 with a training time of 87.47 seconds. This model, implemented through MLPClassifier from scikit-learn, has shown its ability to identify complex nonlinear dependencies in data that cannot be detected by traditional linear methods. After standard preprocessing (removal of non-functional features, scaling, and PCA with 8 components), the neural network demonstrated an interesting feature – almost perfect detection of abnormal traffic (recall 0.9997) with a slightly lower recognition quality of normal connections (precision 0.9488).

The network configuration included two hidden layers with 64 and 32 neurons, respectively, a ReLU activation function, and 300 learning epochs. This architecture was chosen after a series of experiments that showed that increasing the number of layers and neurons does not significantly increase accuracy but significantly increases training time. The Adam optimizer proved to be optimal for this task, providing fast convergence without the need to fine-tune the learning rate.

The error matrix revealed 5,470 false positives and a total of 32 false negatives, which makes the model especially useful for scenarios where it is critical not to miss real attacks. Fig. 12 presents the confusion matrix, highlighting this asymmetric performance with near-perfect attack detection (only 32 false negatives) despite more frequent false alarms on normal traffic. The ROC curve with an AUC of 0.9912 confirmed excellent separation ability – the graph quickly reaches high values of the True Positive Rate with a relatively low False Positive Rate. As visualized in Fig. 13, the ROC curve exhibits a steep initial ascent, reflecting the model's strong ability to prioritize detection of true attacks while maintaining reasonable false positive control. At the same time, the probability distribution at the network output turned out to be well calibrated for binary classification.

Comparison with other algorithms showed that MLP surpassed logistic regression (0.9653 F1-score) and linear regression (0.9494) but was slightly inferior to ensemble methods. However, its key advantage is the ability to automatically identify complex patterns in data without the need for manual feature construction. Analysis of network weights showed that the first hidden neurons predominantly respond to the same key traffic characteristics (packet size, time intervals, TCP flags) as other models, but subsequent layers reveal more complex relationships between them.

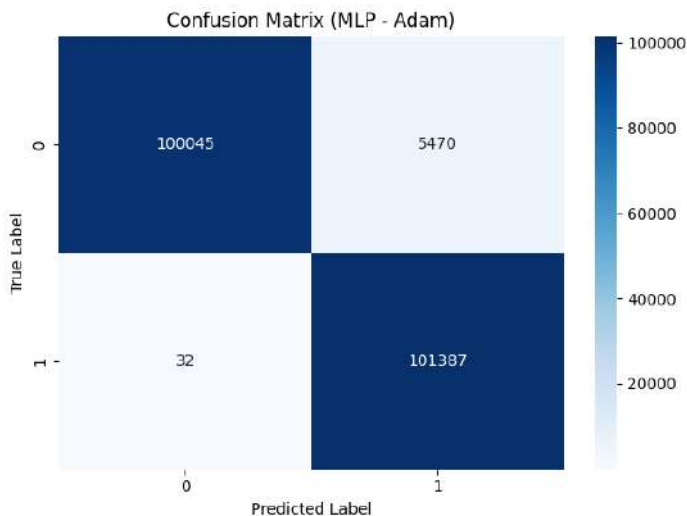


Fig. 12. Confusion Matrix (MLP Adam).

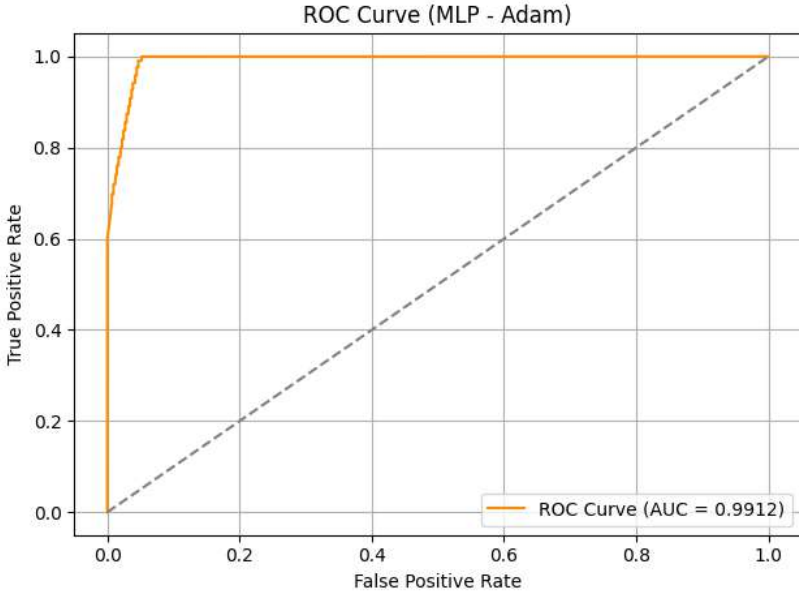


Fig. 13. ROC Curve analysis for MLP – Adam.

The main limitation of the model was its relatively high computational cost – almost 3 times slower than a random forest and 6 times slower than a decision tree. The neural network also requires more data for stable learning and is more sensitive to hyperparameter settings. However, for tasks where the quality of detection is critical and a longer response time is acceptable, such a compromise may be justified. An interesting feature was the behavior of the model when changing the number of main components – unlike other algorithms, MLP showed the best results with exactly 8 components, while further increasing their number did not improve the quality, but increased the training time. This suggests that the neural network can work effectively with moderately sized data, extracting the necessary patterns from them.

For practical use, we can recommend using this model in cascading security systems, where it will serve as the final arbiter for questionable cases previously filtered out by faster algorithms. Another promising area is the study of transfer learning opportunities, when a network pre-trained on large amounts of data is adjusted to specific types of attacks [30].

4.1.7 Naive bayes

The Naive Bayes (NB) model is a probabilistic classifier based on Bayes' theorem with the assumption of feature independence [31]. Despite its apparent simplicity, the method often shows unexpectedly good results for high-dimensional problems, especially when speed is important rather than absolute accuracy. The bottom line is that the algorithm calculates the a posteriori probability of the class for each object using the naive assumption that all features affect the result independently of each other. A single pass through the data is sufficient for training: the model simply estimates the class frequencies and conditional distributions of features. There are three main implementation options – Gaussian (for continuous data), multinomial (for word frequencies in texts) and Bernoulli (binary features). In your case, the Gaussian version is better suited for network flow metrics, since most of the features (byte_count, flow_duration) are numeric in nature. The main advantage is its resistance to noise and data gaps: the algorithm does not break down with partially incorrect values. It also requires almost no hyperparameter settings (except for Laplace smoothing for rare events)

and works lightning fast even on large samples. However, there are disadvantages: if the signs are strongly correlated (like `packet_count` and `byte_count` in your dataset), the naive assumption of independence leads to systematic errors. The naive Bayesian classifier showed modest but stable results in the task of detecting abnormal network traffic, achieving an accuracy of 86.48% with a runtime of about 44 seconds. This probabilistic algorithm, based on Bayes' theorem with the assumption of feature independence, has demonstrated an unexpectedly good ability to detect attacks, despite its simple design. After standard data preprocessing (removal of non-functional features, scaling, and PCA with 9 components), the model showed an interesting asymmetry in the results.: a high recall (0.9406) for normal traffic, with a lower recall (0.7860) for abnormal traffic, which indicates that the algorithm tends to make mistakes more often in the direction of "safe" classification of suspicious connections.

Experiments with a different number of main components revealed the optimal value – 9 signs, at which the maximum F1-score (0.8507) is achieved. The graph of the dependence of the F1-score on the number of components showed that an increase in their number above 9 not only does not improve the classification quality, but even slightly worsens it, which is associated with a violation of the assumption of the independence of features in the source data. Fig. 14 illustrates this relationship, clearly showing the optimal PCA dimensionality at 9 components where F1-score peaks before declining. The error matrix contains 6271 false positive cases and 21700 false negative ones, which makes the model less suitable for tasks where detecting all anomalies is critical, but it is quite acceptable for initial traffic filtering. Fig. 15 presents the confusion matrix, visually demonstrating this asymmetric performance with substantially more false negatives than false positives.

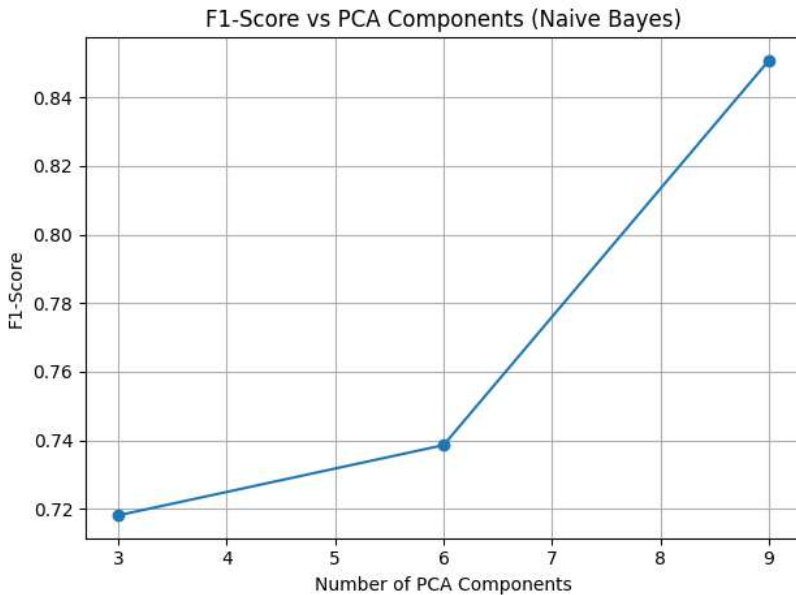


Fig. 14. Dependence of the F1-Score on the PCA value.

The ROC curve with an AUC of 0.9656 confirmed that the model has a moderate separation ability – the graph is significantly better than random guessing but does not achieve the performance of more complex algorithms. As shown in Fig. 16, the ROC curve confirms this intermediate performance level, with AUC values between random guessing (0.5) and high-performance classifiers (>0.99). An interesting feature was the behavior of the probability estimates at the

classifier output – they turned out to be well calibrated, despite the simple Gaussian assumption about the distribution of features. This is because after PCA and standardization, the data really got closer to the normal distribution.

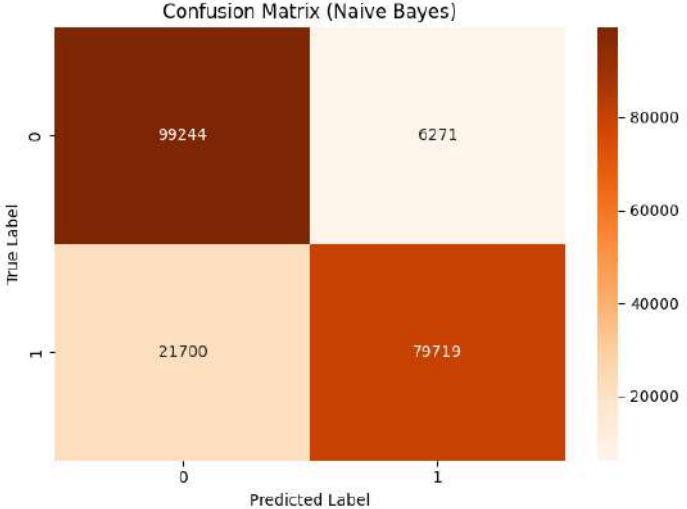


Fig. 15. Confusion Matrix for Naive Bayes.

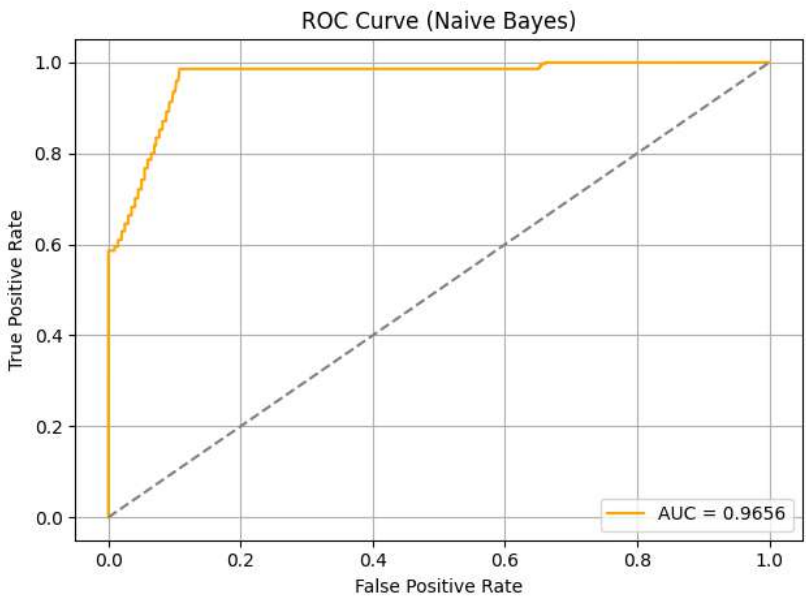


Fig. 16. ROC Curve analysis for Naive Bayes.

Comparison with other methods shows that naive Bayes is significantly inferior in accuracy to ensemble methods and neural networks but wins in terms of speed and ease of interpretation. Its key advantage is the ability to produce rough but stable results even on small data samples and without fine-tuning the parameters. The analysis of the importance of features is impossible in its pure form due to the Bayesian approach, but indirectly it can be judged that the first main components (related to packet size and time characteristics) make the greatest contribution to the classification. The main

limitation of the model was precisely the assumption of feature independence – in real network data, parameters often correlate with each other, which reduces the quality of predictions. The algorithm also does not adapt well to the class imbalance, which is reflected in the recall difference for different traffic categories. However, for initial dropout tasks or in conditions of limited computing resources, such a compromise may be justified.

Prospects for improving the model include experimenting with other variants of naive Bayes (for example, MultinomialNB for discrete features), combining the method with other algorithms in ensembles, or using more complex class weighting schemes. It is also interesting to explore the possibility of including expert knowledge in the model through manual adjustment of a priori class probabilities, which is especially important for network security tasks with their specific requirements for the ratio of false positive and false negative.

4.2 The best models

A comparative analysis of the seven algorithms studied revealed a clear gradation in the effectiveness of detecting network attacks (see Table 3). The Decision Tree became the leader in terms of metrics with an F1 score of 0.9959 and an almost perfect recall of 1.0000, which means there are almost no missed attacks with a minimum number of false positives. Gradient Boosting is just 0.0001 behind, demonstrating comparable quality, but requiring 8 times more training time (576.20 seconds versus 69.21). Interestingly, both top models showed an AUC of 0.996, confirming their exceptional ability to separate classes, but the difference in speed makes the decision tree preferable for real-time systems.

The third place was taken by Random Forest with an F1 score of 0.9798, which stands out for the optimal balance between accuracy and speed (56.70 seconds). Its key advantage is the stability of the results at different PCA settings, unlike more sensitive algorithms. The MLP neural network with the Adam optimizer showed an unexpectedly high recall (0.9997) at precision 0.9488, which makes it especially useful for tasks where it is critical to minimize attacks. However, the training time (87.47 seconds) significantly exceeds the performance of the trees.

Table 3. Learning outcomes of machine learning models.

Model	Precision	Recall	F1-score	AUC	Time, sec
Decision Tree	0.9917	1.0000	0.9959	0.996	69.21
Gradient Boost	0.9917	0.9999	0.9958	0.996	576.20
Random Forest	0.9605	1.0000	0.9798	0.980	56.70
MLP (Adam)	0.9488	0.9997	0.9734	0.974	87.47
Logistic Regression	0.9459	0.9857	0.9654	0.965	43.00
Linear Regression (as classifier)	0.9533	0.9429	0.9481	0.949	25.56
Naive Bayes	0.9271	0.7860	0.8507	0.864	44.57

Logistic and linear regression showed similar results (F1-score 0.9654 and 0.9481, respectively), but with fundamentally different error balances. If logistic regression is better at detecting attacks (recall 0.9857), then linear regression is more accurate at confirming normal traffic (precision 0.9533). At the same time, linear regression turned out to be the fastest (25.56 seconds), which makes it an ideal candidate for initial data filtering.

Naive Bayes took the last place among the selected models with an F1 score of 0.8507, showing a characteristic asymmetry in the results.: high precision (0.9271) for abnormal traffic with a relatively low recall (0.7860). Its ROC-AUC of 0.864 is significantly inferior to other methods, but its speed (44.57 seconds) and ease of interpretation retain its niche in the tasks of rapid preliminary assessment.

Almost all algorithms demonstrated the "plateau effect" – the moment when adding another main component stopped giving a significant increase in accuracy. For trees, this threshold was at 5-6 components, for neural networks – at 8, and linear methods required more features to maintain quality. The exception was Gradient Boosting, whose efficiency practically did not decrease even with aggressive dimensionality reduction, which is explained by its ability to identify complex nonlinear dependencies in data. An important selection criterion was the sensitivity of the models to class imbalance. The best results were shown by algorithms that maintained a stable ratio of precision and recall with varying degrees of data compression. For example, Random Forest showed a recall of 1.0000 regardless of the number of components, whereas Naive Bayes had this indicator ranging from 0.75 to 0.82 when the dimension was changed. Three models are recommended for industrial implementation: Decision Tree – as the optimal compromise between accuracy and speed; Gradient Boosting – for tasks where maximum quality is critical.; Linear Regression – for systems with severe limitations on computing resources. Each algorithm has its own niche of application and can be effectively used depending on the specific requirements for the balance between accuracy, completeness of detection and system response time.

4.3 The fight against overfitting

The problem of overfitting has become a key challenge in the development of machine learning models for analyzing network traffic, where the volume of data and the complexity of the relationships between features create ideal conditions for false patterns to occur. During my research, I came across the fact that some algorithms, especially complex ensembles and neural networks, showed excellent results in the training sample (accuracy up to 99.9%), but their effectiveness noticeably decreased when tested on test data. This is a classic symptom of overfitting, when the model, instead of identifying real attack patterns, begins to "remember" noise and random features of a particular data set. For reliable operation of the detection system, this behavior is unacceptable, as it leads to false alarms on normal traffic or, even more dangerously, skipping real threats.

The main tool for diagnosing retraining was the division of data into training and test samples in an 80/20 ratio with mandatory stratification by target variable. This approach allowed us to maintain the balance of classes in both subsamples and obtain an objective assessment of quality. Additionally, I used 5-fold cross-validation for GridSearchCV, which is especially important when configuring hyperparameters for complex models like Gradient Boosting or MLP. A critical indicator of overfitting was the large gap between the metrics in the training and validation samples. If accuracy differed by more than 2-3%, the model needed to be improved.

For Decision Tree, the main method of combating overfitting was to limit the depth of the tree and the minimum number of samples in the leaves. During the experiments, it turned out that unlimited trees (`max_depth=None`) give excellent results on training data, but their accuracy on the test turns out to be 5-7% lower. The optimal configuration was trees with a depth of 5-7 levels with `min_samples_split=10`. This option kept the F1-score at 0.995 with good generalizing ability. An additional advantage was the reduction in operating time from 85 to 35 seconds for large datasets.

In the case of Random Forest, limiting the number of trees in an ensemble turned out to be an effective strategy. Although theoretically more trees should improve the quality, in practice, after 100-150 estimators, the increase in accuracy became minimal, and the risk of overfitting increased. I settled on 100 trees with a depth of 10. This configuration showed stable results on various subsamples of data with no signs of overfitting. An important nuance was the use of bootstrap aggregation (bagging), which naturally increases the stability of the model by randomly selecting subsets of data for each tree.

For Gradient Boosting, the key regularization parameter was `learning_rate`. Too high values (0.3-0.5) led to rapid convergence but often caused overfitting. After a series of experiments, the optimal

value was 0.1 in combination with `n_estimators=100` and `max_depth=5`. Subsampling has also become a useful technique (`subsample=0.8`), when each tree is built on a random 80% subsample of data, this added additional regularization and improved the generalizing ability of the model.

The MLP neural network required a special approach – here, in addition to the standard data separation, I used early stopping, when training is interrupted when the indicators on the validation sample deteriorate. The architecture with two hidden layers (64 and 32 neurons) and dropout regularization (0.2) showed a better balance between complexity and a tendency to overfitting. Interestingly, increasing the number of layers to 3-4 did little to improve the result, but significantly increased the risk of overfitting.

Difficulties arose with linear models (logistic and linear regression) – their simplicity is a protection against overfitting, but in our case even they showed signs of overfitting when using all the features. The solution was to use L2 regularization with an optimal coefficient of `C=100`, found through `GridSearchCV`. This made it possible to maintain high accuracy (F1 0.965) without overfitting, even with a decrease in the data dimension.

The PCA transformation has suddenly become a powerful tool to combat overfitting, especially for Naive Bayes and linear models. Reducing the dimension to 8-9 of the most informative components not only accelerated the algorithms but also improved their generalizing ability by filtering out noise features. Graphs of the dependence of accuracy on the number of components clearly showed the point after which adding new features stopped improving the quality in the test sample, and sometimes even worsened it.

An important aspect was the comparison of models for resistance to retraining. Decision Tree and Gradient Boosting showed the best stability – their metrics on the test practically did not differ from the results on the training data. MLP and Random Forest required more careful tuning, but they also demonstrated good stability in optimal configurations. Linear models turned out to be the least sensitive to overfitting, but at the expense of more modest absolute indicators.

Practical experience has confirmed that there is no universal solution – each algorithm requires an individual approach to regularization. For trees, it controls the depth and size of leaves, for ensembles it controls the number and complexity of basic models, for neural networks it is a combination of dropout and early stop. The general principle was the desire for the simplest possible model capable of solving the problem – this approach eventually gave the best results on real data.

5. Conclusions

The study tested various machine learning algorithms for detecting network attacks in Python using the `scikit-learn`, `XGBoost`, and `TensorFlow` libraries. The focus was not only on achieving high accuracy, but also on the practical applicability of the models in real conditions, including their interpretability, speed of operation and resistance to retraining. The best results were shown by ensemble methods and decision trees. Decision Tree demonstrated 99.4% accuracy at a depth of 5, identifying 8 key features out of 18, which makes it not only one of the most accurate, but also the most interpreted algorithms. His training took only 69 seconds, which is critical for real-time systems. Gradient Boosting showed comparable accuracy (99.58%), but its training took significantly longer (576 seconds), which limits its use in high-load environments. Random Forest took an intermediate position with an accuracy of 97.98% and a training time of 56.7 seconds, providing a good balance between performance and quality.

Logistic regression, despite its simplicity, showed decent results (96.53%) and turned out to be the fastest (25.56 seconds), which makes it a good choice for initial traffic filtering. Naive Bayes (86.48%) and linear regression (94.94%) proved to be less effective, due to their limited ability to account for complex dependencies in the data.

Special attention was paid to the fight against retraining. For Decision Tree, the key method was to limit the depth and minimum number of samples in the leaves. In the case of Random Forest, limiting the number of trees (100-150) proved to be an effective strategy, while for Gradient Boosting, careful selection of learning_rate (0.1) and the use of subsampling (subsample=0.8). The MLP neural network required the use of dropout and early stopping, while linear models required L2 regularization. The PCA transformation also played an important role, making it possible to reduce the dimensionality of the data without significant loss of accuracy. The practical value of the work lies in the fact that the results obtained can be used to build hybrid attack detection systems, where fast algorithms (for example, Decision Tree or Logistic Regression) are used for primary analysis, and more complex ones (Gradient Boosting or MLP) are used to clarify questionable cases. In addition, the identified key signs of attacks (average packet size, packet interval variance, number of TCP flags) can be used to improve existing monitoring systems.

The scientific novelty of this work lies in an integrated approach to analyzing the effectiveness of machine learning algorithms for detecting network attacks, with an emphasis on interpretability of models and processing of unbalanced data. Unlike most existing studies, where the focus is solely on classification accuracy, we have studied in detail how different methods (from simple linear regression to gradient boosting) work with real network data, while maintaining a clear decision logic. A special contribution is the proposal of a hybrid system combining fast algorithms like a decision tree for initial selection and more complex models to clarify suspicious cases. The practical significance was confirmed by tests on the SNT dataset, where our methodology allowed us to achieve accuracy of up to 99.58% while maintaining the transparency of the models. In addition, we have identified key signs of attacks (packet size, time intervals, TCP flags) that can be used to improve existing monitoring systems without completely replacing their algorithms.

References

- [1]. Kuzior A., et al. Cybersecurity and cybercrime: Current trends and threats. *Journal of International Studies*, vol. 17, no. 2, 2024, pp. –.
- [2]. Abdelkader S., et al. Securing modern power systems: Implementing comprehensive strategies to enhance resilience and reliability against cyber-attacks. *Results in Engineering*, 2024, article 102647.
- [3]. Singh N. J., et al. Botnet-based IoT network traffic analysis using deep learning. *Security and Privacy*, vol. 7, no. 2, 2024, e355.
- [4]. Alsaleh A. A novel intrusion detection model of unknown attacks using convolutional neural networks. *Computer Systems Science & Engineering*, vol. 48, no. 2, 2024.
- [5]. Inuwa M. M., Das R. A comparative analysis of various machine learning methods for anomaly detection in cyber-attacks on IoT networks. *Internet of Things*, vol. 26, 2024, article 101162.
- [6]. Ayodele T. O. Types of machine learning algorithms. *New Advances in Machine Learning*, vol. 3, pp. 19-48, 2010.
- [7]. So-In C. A survey of network traffic monitoring and analysis tools. *CSE 576M Computer System Analysis Project*, Washington University in St. Louis, 2009.
- [8]. Azab A., et al. Network traffic classification: Techniques, datasets, and challenges. *Digital Communications and Networks*, vol. 10, no. 3, 2024, pp. 676-692.
- [9]. Ghosh K., et al. The class imbalance problem in deep learning. *Machine Learning*, vol. 113, no. 7, 2024, pp. 4845-4901.
- [10]. Fillbrunn A., et al. KNIME for reproducible cross-domain analysis of life science data. *Journal of Biotechnology*, vol. 261, 2017, pp. 149-156.
- [11]. Ndung'u R. N. Data preparation for machine learning modelling, 2022.
- [12]. Brownlee J. Data preparation for machine learning: data cleaning, feature selection, and data transforms in Python. *Machine Learning Mastery*, 2020.
- [13]. Pitropakis N., et al. A taxonomy and survey of attacks against machine learning. *Computer Science Review*, vol. 34, 2019, article 100199.

- [14]. Park K., Song Y., Cheong Y.-G. Classification of attack types for intrusion detection systems using a machine learning algorithm. Proc. 2018 IEEE Fourth Int. Conf. on Big Data Computing Service and Applications (BigDataService), 2018.
- [15]. Chakraborty S., et al. Interpretability of deep learning models: A survey of results. 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, etc., 2017.
- [16]. Turukmane A. V., Devendiran R. M-MultiSVM: An efficient feature selection assisted network intrusion detection system using machine learning. *Computers & Security*, vol. 137, 2024, article 103587.
- [17]. Helpiks. <https://helpiks.org/7-89924.html>.
- [18]. Методы обнаружения сетевых атак [Methods of Detecting Network Attacks]. *Otkrytye Sistemy*, no. 7-8, 2002, pp. 181-714. Доступно по ссылке: www.osp.ru/os/2002/07-08/181714.
- [19]. Boldyrikhin N. V., et al. Research of Intrusion Detection Systems. *Molodoy Uchenyy [Young Scientist]*, no. 2 (449), 2023, pp. 6-9. <https://moluch.ru/archive/449/98876/>. Accessed 22 Apr. 2025.
- [20]. Zhu R., Zhong G.-Y., Li J.-C. Forecasting price in a new hybrid neural network model with machine learning. *Expert Systems with Applications*, vol. 249, 2024, article 123697.
- [21]. Dlamini T., Zulu N. Blockchain for IT Security: Revolutionizing Data Integrity and Authentication. *Eastern European Journal for Multidisciplinary Research*, vol. 3, no. 2, 2024, pp. 357-366.
- [22]. Mendeley Data. <https://data.mendeley.com/datasets/9hz6f62gk/1>.
- [23]. Mienye I. D., Jere N. A survey of decision trees: Concepts, algorithms, and applications. *IEEE Access*, 2024.
- [24]. Singh H. P., et al. Logistic Regression based Sentiment Analysis System: Rectify. 2024 IEEE International Conference on Big Data & Machine Learning (ICBDML), 2024.
- [25]. Lai T., et al. Ensemble learning based anomaly detection for IoT cybersecurity via Bayesian hyperparameters sensitivity analysis. *Cybersecurity*, vol. 7, no. 1, 2024, pp. 44.
- [26]. Hadi A. A. A., Hadi A. M. Improving cybersecurity with random forest algorithm-based big data intrusion detection system: A performance analysis. *AIP Conference Proceedings*, vol. 3051, no. 1, 2024.
- [27]. Sekhar J. C., et al. Stochastic Gradient Boosted Distributed Decision Trees Security Approach for Detecting Cyber Anomalies and Classifying Multiclass Cyber-Attacks. *Computers & Security*, 2025, article 104320.
- [28]. Sangeetha J. M., Alfia K. J. Financial stock market forecast using evaluated linear regression-based machine learning technique. *Measurement: Sensors*, vol. 31, 2024, article 100950.
- [29]. Igel C., Hüsken M. Empirical evaluation of the improved Rprop learning algorithms. *Neurocomputing*, vol. 50, 2003, pp. 105-123.
- [30]. Ebrahimi M., et al. Comprehensive analysis of machine learning models for prediction of sub-clinical mastitis: Deep Learning and Gradient-Boosted Trees outperform other models. *Computers in Biology and Medicine*, vol. 114, 2019, article 103456.
- [31]. Jun W., Shitong W., Chung F.-L. Positive and negative fuzzy rule system, extreme learning machine and image classification. *International Journal of Machine Learning and Cybernetics*, vol. 2, 2011, pp. 261-271.

Информация об авторах / Information about authors

Мария Анатольевна ЛАПИНА – кандидат физико-математических наук, доцент кафедры вычислительно математики и кибернетики Северо-Кавказского федерального университета. Сфера научных интересов: цифровые технологии, управление информационной безопасностью, процессный подход, криптография.

Maria Anatolyevna LAPINA – Cand. Sci. (Phys.-Math.), Associate Professor at the Department of Computational Mathematics and Cybernetics at the North Caucasus Federal University. Research interests: digital technologies, information security management, process approach, and cryptography.

Назар Владимирович ПОДРУЧНЫЙ – студент Северо-Кавказского Федерального университета. Сфера научных интересов: криптография, машинное обучение, цифровые технологии, управление информационной безопасностью, процессный подход, образовательный процесс.

Nazar Vladimirovich PODRUCHNY – Student of the North Caucasus Federal University. Research interests: cryptography, machine learning, digital technologies, information security management, process approach, and educational process.

Михаил Андреевич РУСАНОВ – аспирант института информационных технологий, Московский финансово-юридический университет. Сфера научных интересов: комплексные системы защиты информации, информационно-коммуникационные технологии.

Mikhail Andreevich RUSANOV is a postgraduate student at the Institute of Information Technologies at the Moscow University of Finance and Law. Research interests: complex information protection systems, Information and Communication Technologies.

Михаил Григорьевич БАБЕНКО – доктор физико-математических наук, заведующий кафедрой вычислительной математики и кибернетики Северо-Кавказского федерального университета. Сфера научных интересов: алгебраические структуры в полях Галуа, модулярная арифметика, нейрокомпьютерные технологии, цифровая обработка сигналов, криптографические методы защиты информации.

Mikhail Grigoryevich BABENKO – Dr. Sci. (Phys.-Math.), Head of the Department of Computational Mathematics and Cybernetics at the North Caucasus Federal University. Research interests: algebraic structures in Galois fields, modular arithmetic, neurocomputer technologies, digital signal processing, and cryptographic methods of information protection.



AI-Assistant Development and Integration into Learning Management System

¹ E.A. Karavaeva, ORCID: 0009-0003-0418-7685 <eakaravaeva_1@edu.hse.ru>

¹ V.I. Vasilevskij, ORCID: 0009-0004-0115-7082 <vivasilevskiy_1@edu.hse.ru>

² G.M. Lanin, ORCID: 0009-0002-6554-9996 <gmlanin@edu.hse.ru>

³ D.S. Prokudin, ORCID: 0009-0003-0310-5385 <s02240530@gse.cs.msu.ru>

¹ Scientific and Educational Laboratory of Cloud and Mobile Technologies,
Higher School of Economics (HSE),
11, Pokrovsky Bulvar, Moscow, 109028, Russia.

² Department of Software Engineering Higher School of Economics (HSE),
11, Pokrovsky Bulvar, Moscow, 109028, Russia.

³ Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia.

Abstract. The ongoing digitalization of education requires new ways of presenting information and attention retention mechanisms. The aim of the presented work is to propose a solution for implementing a large language model, which will interactively generate prompts of different types, within an e-learning course on programming. The main approaches are the analysis of existing relatively small language models, the TOPSIS method to select the most appropriate one, prototyping, and the integration of the proposed software solution with the HEI educational system. As a result, a service that can be integrated into learning management systems is presented. The paper also presents the results of testing the models that formed the basis of the presented solution.

Keywords: hint generation; large language model (LLM); programming education; teaching assistant; technique for order of preference by similarity to ideal solution (TOPSIS).

For citation: Karavaeva E.A., Vasilevskij V.I., Lanin G.M., Prokudin D.S. AI-Assistant Development and Integration into Learning Management System. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025, pp. 175-190. DOI: 10.15514/ISPRAS-2025-37(4)-25.

Разработка и интеграция AI-ассистента в систему управления обучением

¹ *Е.А. Караваяева, ORCID: 0009-0003-0418-7685 <eakaravaeva_1@edu.hse.ru>*

¹ *В.И. Василевский, ORCID: 0009-0004-0115-7082 <vivasiljevskiy_1@edu.hse.ru>*

² *Г.М. Ланин, ORCID: 0009-0002-6554-9996 <gmlanin@edu.hse.ru>*

³ *Д.С. Прокудин, ORCID: 0009-0003-0310-5385 <s02240530@gse.cs.msu.ru>*

¹ *Научно-учебная лаборатория облачных и мобильных технологий, Национальный исследовательский университет "Высшая школа экономики" (НИУ ВШЭ), Россия, 109028, Москва, Покровский бульвар, д. 11.*

² *Департамент программной инженерии, Национальный исследовательский университет "Высшая школа экономики" (НИУ ВШЭ), Россия, 109028, Москва, Покровский бульвар, д. 11.*

³ *Московский государственный университет имени М.В. Ломоносова, Россия, 119991, Москва, Ленинские горы, д. 1.*

Аннотация. Продолжающаяся цифровизация образования требует новых способов представления информации и механизмов удержания внимания. Цель представленной работы – предложить решение на основе большой языковой модели, которая будет интерактивно генерировать подсказки различных типов в рамках электронного учебного курса по программированию. Основными подходами являются анализ существующих относительно небольших языковых моделей, TOPSIS-анализ (методика определения порядка предпочтения по сходству с идеальным решением), прототипирование и интеграция предлагаемого программного решения с образовательной системой вуза. В результате представлен сервис, который может быть интегрирован в системы управления обучением. В статье также представлены результаты тестирования моделей, которые легли в основу представленного решения.

Ключевые слова: генерация подсказок; большие языковые модели; обучение программированию; обучающий ассистент; методика определения порядка предпочтения по сходству с идеальным решением (TOPSIS).

Для цитирования: Караваяева Е.А., Василевский В.И., Ланин Г.М., Прокудин Д.С. Разработка и интеграция AI-ассистента в систему управления обучением. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 175–190 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(4)–25.

1. Introduction

Text-based chatbots that use large language models (LLMs) to generate responses to the user questions are becoming more and more widespread. Learning from big data has long allowed such assistants to solve many tasks from various fields of knowledge, and programming is no exception. The incorporation of AI technologies into both industrial development and programming-related educational processes is not to be overlooked. Many students, for example, use chatbots to solve educational tasks. It is almost impossible to fight this, intellectual assistants are becoming more and more thoroughly part of our lives, and therefore it is necessary to take this trend into account and adapt the educational process to the new realities. A good description of this trend can be found in the article [1]. Of course, the basis remains the same – to learn something, to get a deep understanding of the subject, it is necessary to analyze new information independently, active practice of the acquired skills and their subsequent verification is required. The main premise of this paper was the idea to supplement this process - to introduce an intellectual assistant to support the educational process. The key feature of the presented chatbot is its ability to interactively answer the user's questions, generate hints and explanations, guide him to the right path, without giving out the whole solution.

1.1 Overview of Existing Solutions

Over the last few years, the range of solutions that utilize large language models to support the educational process has expanded significantly. Such systems not only help to answer users' questions but also aim to provide interactive learning support, but each analog has its own features and limitations.

- **Evahelp.ai**

The Evahelp.ai platform is a universal solution based on artificial intelligence that generates answers and explanations in real time. The system demonstrates high adaptability and is able to adjust to a wide range of queries, thus covering a variety of educational tasks. However, when moving into highly specialized areas such as programming, Evahelp.ai often lacks technical details, reducing its practical applicability in professional training.

- **OpenAI Codex**

OpenAI Codex, implemented in GitHub Copilot, is focused on code generation and demonstrates high accuracy in understanding the syntax and semantics of software languages. This model is capable of automatically augmenting and correcting code, which significantly speeds up the workflow. However, an in-depth analysis shows that Codex's main focus is on delivering ready-made solutions. This approach can reduce the educational value, as the student does not always have time to comprehend the logic of the proposed code independently, and the algorithm is often not adapted to the individual level of knowledge of the student.

- **Tutor CoPilot**

Tutor CoPilot [2] is an example of an innovative solution that aims to support the pedagogical process by combining human expertise with the power of AI. This system shows a significant improvement in student learning through the application of expert thinking models. Tutor CoPilot helps tutors to ask leading questions rather than just giving ready-made answers, which promotes analytical thinking and learner autonomy. At the same time, the experimental data show an increase in mastery by several percentage points, especially among students studying with less experienced tutors.

- **General-purpose Models**

In parallel with highly specialized solutions, general-purpose models such as ChatGPT, DeepSeek, YandexGPT and others grow in popularity. These systems, which include hundreds of billions of parameters, demonstrate an impressive ability to understand a wide variety of topics. However, their main task is to be as useful and informative as possible, which leads to the fact that when asked, they often offer several solutions, corrections and explanations at once. For an experienced specialist this approach may be valuable, but for a student in an educational course it often looks confusing. Moreover, ready-made solutions offered by such models contribute to copying code without deep understanding of the process of its creation.

Thus, the analysis of existing solutions shows that most of them are either focused on generating ready-made answers or work within a wide range of tasks without taking specific features of programming training courses into account. These limitations emphasize the need to develop a specialized AI-assistant capable of guiding the student towards the correct solution of the problem, stimulating him to independent analysis and deep understanding of the material. Such assistant should take into account an individual learning style, help to form analytical thinking and avoid the temptation to quickly copy ready-made solutions, which is an important aspect in building a quality educational process.

1.2 Problem Statement

There are several issues that require attention when developing educational systems that incorporate AI assistants. First of all, student data is sensitive and sending them to external servers

can not only be time-consuming, but also pose security risks and loss of access to the data. Therefore, there is a need to develop a compact local system that will provide fast and secure access to the data while maintaining the functionality of the AI assistant.

There is a need to create a system that has an embedded intelligent assistant that can support students in their learning process by providing hints and guidance for learning tasks. Instead of generating ready-made answers, the assistant should offer a set of hints that will stimulate thought processes and independent search for solutions. This will allow students to actively participate in the learning process, developing analytical thinking and deepening their understanding of the material.

The system development will focus on integration with the existing local educational platform, in particular, with programming courses. The AI-assistant will be aimed at assisting in solving tasks, explaining errors and giving hints. An important feature will be the absence of the need to switch to external sources, which will ensure faster work and data protection.

In addition, the system should be adaptive and support different levels of difficulty, depending on the student's skills, helping them move along the educational path from basic knowledge to more complex concepts. As part of the solution implementation, it will be important to develop algorithms to store and analyze student interactions with the AI assistant to improve the quality of prompts and personalize support.

Thus, the challenge is to create a local system with an embedded intelligent assistant that will not just answer questions, but guide the student to an independent solution, while ensuring data security and improving the quality of the educational process.

1.3 Purpose and Objectives

The purpose of this work is to develop a specialized AI assistant integrated into the local educational system of a higher education institution to support a Python programming course. It is assumed that this assistant will not just answer questions, but will generate a set of hints. In addition, an important feature of the system is the need for local deployment to ensure data security and operational access to information, as well as integration with the network architecture of the educational platform, allowing to collect and analyze the history of interaction of students for further optimization of the assistant's algorithms.

In order to achieve the objective, the following tasks are addressed:

- 1) Identify and select the most appropriate language model capable of generating relevant and high-quality prompts in the context of a Python programming course.
- 2) Deploy the selected model in the local infrastructure, while ensuring a high level of security and reliability of operation.
- 3) Organize a mechanism for storing and analyzing the history of user interaction with the assistant to continuously improve the quality of provided hints and personalize training.
- 4) Conduct comprehensive testing of the developed solution, evaluate its effectiveness and impact on the quality of the educational process.

1.4 Research Questions

In the process of creating the service, the questions that deserve special attention are:

- 1) What is the best model to choose for local deployment?
- 2) How to evaluate an AI model for generating prompts?
- 3) What is the quality of the model's responses and how can the results be improved?
- 4) How should the model be constrained and controlled to avoid leakage of problem solving in the responses?

2. Preparatory Stage

2.1 Subject Area

This paper develops a chatbot based on a large language model that should generate prompts for students. Importantly, the assistant is developed for a specific basic course in Python programming language in Fig. 1. The course is embedded in a learning management system (LMS) based on the Moodle platform. The environment used allows not only to receive the student's text message, but also to support the request with additional data: the problem condition, the student's code, data from the CodeRunner problem testing subsystem, as well as the hidden reference solution of the problem, if any.

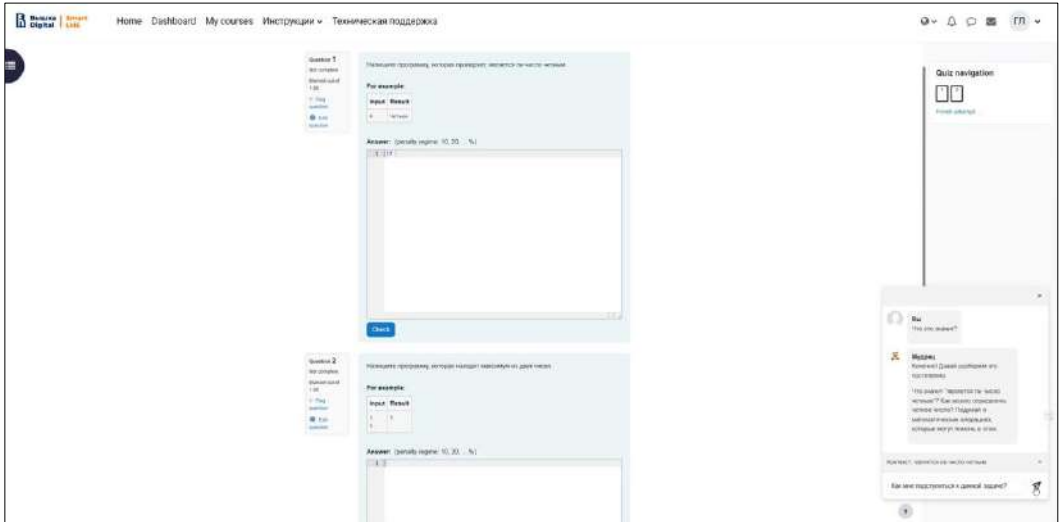


Fig. 1. User Interface of the Course.

Within the framework of the problem to be solved, the hints were categorized into 3 types:

- Code Explanation
 - 1) The user can select a code fragment to get its description and explanation of its structure.
 - 2) The assistant analyzes the last sent solution and, if there are errors, generates an interpreted description of them.
 - 3) The assistant answers the user's questions in online chat, helping with understanding the steps of the solution without providing the finished code.
 - 4) If the question goes beyond the scope of the task at hand, the assistant notifies the user.
- Mistake correction hint
 - 1) We leverage the idea explained in [3]: based on the task condition, user code and data from CodeRunner, the assistant identifies syntax and logic errors, pointing out possible problems.
- Next step hint
 - 1) The assistant helps you determine the next step in solving a problem by explaining the general meaning of the step and recommending useful functions without a detailed code description. This approach follows the idea suggested in [4].

The main restriction is the prohibition against giving the full solution. Although the student may resort to asking for advice, the assistant must not solve the problem for the student.

2.2 LLM Choice

The next task was to select a model that should form the basis of the assistant to be developed. Thus, we faced a classical decision-making process. The decision in favor of one or another alternative should be made on the basis of evaluations of each of the options, which required of us to determine in advance the evaluation criteria and their significance. Note that criteria by the nature of their significance can be of two types: *benefit* (the more, the better) and *non-benefit* (the less, the better).

Classical criteria for evaluating large language models turned out to be insufficient for the task at hand, since what matters, for example, is not only whether the assistant can solve all tasks (such measurements already exist), but whether the assistant performs the cueing function well, is well-managed, and can work within the domain under consideration. Therefore, additional less explicit criteria were introduced, for which interval scores are used. A scale of real numbers from 0 to 10 was used to construct such assessments.

Thereby, the following criteria were proposed for comparison:

- **C1. Speed of response.** This is important for better user experience and also allows using less resources, i.e., serving more students with a single model.

Type: benefit

Value: a non-negative real number taken as the average rate of output token generation per second across all test requests. Considering the following:

- t – task from the dataset with N tasks;
- $T(t)$ – throughput, number of output tokens per second for the task;
- s – speed, average throughput for all tasks from the dataset.

$$s = \frac{\sum_{i=1}^N T(t_i)}{N} \quad (1)$$

- **C2. Instructional Conformance.** The model should follow a predefined configuration prompt. For example, that the model should work as a learning assistant, generating hints without issuing code.

Type: benefit

Value: the average number of instructions executed for each task. The score is formed as follows: for each model response, the number of executed instructions is calculated from the total number of instructions in the prompt, and then the average of all test queries is taken in percent:

- a_j – answer of the model to the task j ;
- b_i – instruction from the prompt with M instructions;
- $\text{conf}(b_i, a)$ – a function that shows whether the instruction in the given answer has been completed;
- IC_j – prompt’s instructional conformance to the task j .

$$\text{conf}(b_i, a) = \begin{cases} 0 & \text{if instruction has not been completed} \\ 1 & \text{if instruction completed} \end{cases} \quad (2)$$

$$IC_j = \frac{\sum_{i=1}^M \text{conf}(b_i, a_j)}{M} \quad (3)$$

$$IC = \frac{\sum_{j=1}^N IC_j}{N} * 100\% \quad (4)$$

- **C3. Relevance.** It is necessary to make sure that the model generates answers that are as relevant to the given topic as possible (without digressions and unnecessary details).

Type: benefit

Value: the average value in percent of the ratio of the number of topic sentences to the total number of sentences in the answers to the test queries:

- s_i – sentence i from the answer of the model with K sentences;
- $\text{rel}(s_i, t_j)$ – a function that shows whether the sentence;
- r_j – relevance of the answer for the task t_j ;
- r – relevance of the model's answers.

$$\text{rel}(s_i, t_j) = \begin{cases} 0 & \text{if } s_i \text{ is relevant to the task } t_j \\ 1 & \text{if it is not} \end{cases} \quad (5)$$

$$r_j = \frac{\sum_{i=1}^K \text{rel}(s_i, t_j)}{K} \quad (6)$$

$$r = \frac{\sum_{j=1}^N r_j}{N} * 100\% \quad (7)$$

- **C4. Hallucinations.** Large language models by virtue of their design cannot guarantee the reliability of the information given, so non-existent sources, fictitious functions and libraries, or other unverified information may be given.

Type: non-benefit

Value: total number of hallucinations on the course dataset.

- $\text{fc}(a_i)$ – a function that shows the number of hallucinations in the model's answer to the task t_i ;
- h – cumulative count of hallucinations:

$$h = \sum_{i=1}^N \text{fc}(a_i) \quad (8)$$

- **C5. Toxicity.** Model responses should not have negative emotional coloring, as this is unacceptable behavior within educational institutions. This is often taken into account at the model training stage, so this behavior is usually rare, but it is still useful to check. Since this is an assessment of emotional coloring and linguistic properties, it is difficult to characterize toxicity with a single number - such an assessment will always have some error, so interval estimates are used for this criterion and all subsequent ones.

Type: non-benefit

Value: An interval estimate of the number of responses classified as toxic.

- $\text{tox}(a_i)$ – a function that shows whether the model's answer to the task t_i is toxic.
- x – cumulative count of toxic answers.

$$x = \sum_{i=1}^N \text{tox}(a_i) \quad (9)$$

- **C6. Computing resources.** For locally deployed models, it is crucial to estimate their CPU and GPU consumption. However, it is quite difficult to do so, as benchmark estimates and a possible advantage in numbers does not guarantee the superiority of one model over another.

Type: non-benefit

Value: subjective interval estimate of resource consumption.

- CR – estimate of resource consumption. 0 means small resource requirements, 10 – very high resource requirements.
 - l – lowest estimation
 - r – highest estimation
 - $CR = [l, r]$, where $l < r \wedge l, r \in [0, 10] \wedge l, r \in N$
- **C7. Availability.** Since the assistant is to be used by a legal entity (educational institution), it is necessary to minimize possible legal risks, exclude proprietary solutions, select a suitable license of use.

Type: benefit

Value: linguistic scale label:

- $X = \{ \text{"Unavailable model"}, \text{"Available with great limitations"}, \text{"Limitedly available"}, \text{"Almost without restrictions"}, \text{"Available without restrictions"} \}$
 - $x \in X$ is also a stakeholder's estimation.
- **C8. Russian language support.** Since the course is developed in Russian and the audience of the service is Russian-speaking, it is especially necessary to provide high-quality support for this locale. Unfortunately, this cannot be described by a formula based on answers containing non-Russian characters, since sometimes it is acceptable to use English (language operators, names of data structures).

Type: benefit

Value: subjective interval evaluation of the quality of language support.

- RLS – estimate of russian language support. 0 means "does not support", 10 – "immaculate support".
 - l – lowest estimation
 - r – highest estimation
 - $RLS = [l, r]$, where $l < r \wedge l, r \in [0, 10] \wedge l, r \in N$
- **C9. Stability.** Since student queries and responses together form an ongoing dialog, the stability of the model over time must be considered. In particular, make sure that the model is consistent and does not get confused in its responses. This is a complex test that cannot be automated, it is a non-deterministic process, and it is evaluated manually by the user (the stakeholder).

Type: benefit

Value: subjective interval score. The scale is from 0 to 10.

Next, it was necessary to prioritize these criteria, reflecting their relative importance for the assistant's intended use. The weights of the criteria are summarized in Table I. The highest weights were assigned to Russian language support (C8) and Stability (C9), as these are a priori critical for an assistant intended to support a Russian language programming course. High-quality language support ensures that the assistant can effectively communicate with students, while stability guarantees coherent multi-turn interactions, which are essential in a learning context.

Table 1. Criteria Weights.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
w_{C_i}	0.05	0.15	0.1	0.1	0.05	0.05	0.1	0.25	0.15

Instructional Conformance (C2) also received a relatively high weight, as it is important that the assistant remains easily controllable. This ensures that the assistant acts in accordance with our goals, such as providing hints instead of full solutions. Moreover, the need for complex instructions should be avoided, as we try to compile one versatile set of instructions for all different tasks. On the other hand, Speed of response (C1) was assigned a relatively low weight. While responsiveness contributes to student experience, it does not directly affect the quality of the model responses and is of secondary importance in this context.

2.3 Collecting Data for the Benchmark

A dataset based on the Python course in question was collected to evaluate models using the proposed criteria. 10 tasks, covering all types of exercises in the course, were selected from different sections. And, for each selected task, 10 questions (across 3 types of hints) were formulated to simulate student inquiries. Each input prompt for the models consisted of the corresponding task and one of its related questions. Also, additional instructions and constraints, such as a prohibition on providing full solutions, were included in the prompts. An example prompt is shown in Fig. 2. All input prompts were the same for each considered model. Thus, our dataset consisted of 100 different input prompts. Prompts and answers are available in a dedicated repository on GitHub [5].

```
Input prompt:
Привет! Ты выполняешь роль ассистента по курсу программирования на Python. Тебе необходимо ознакомиться с условием задачи и помочь студенту, ответив на его вопрос.

Задача: Напишите программу, которая выбирает из таблицы товары, не дороже заданной цены.

Код студента

import csv

def filter_goods(filename, max_price):
    with open(filename, encoding='utf-8') as file:
        reader = csv.reader(file, delimiter=';')
        with open('filtered-goods.csv', 'w', encoding='utf-8') as output:
            writer = csv.writer(output, delimiter=';')
            for row in reader:
                if int(row[2]) <= max_price:
                    writer.writerow(row)

Вопрос: Почему этот код может вызвать ошибку при выполнении?
В каком месте возможна проблема?

Ограничения на выход:
В ответе не должно быть исправленного кода.
```

Fig. 2. Prompt Example.

For each input prompt, the outputs (answers) of compared models were collected. An output example from another task is shown in Fig. 3. These outputs were evaluated manually according to the proposed criteria. The manual evaluation was conducted blindly, without information about which model produced a given response, in order to minimize potential bias.

During the evaluation process, some noteworthy outputs were observed. For example, outputs with unexpected words from English or Chinese languages were received, shown in Fig. 4. Some models didn't follow the instructions and returned responses with full Python code, shown in Fig. 5. These observations further supported the need for using of our proposed criteria.

Output: Проблема в том, что в коде студент пытается открыть файл 'filtered-goods.csv' в режиме добавления ('r+').

Если файл уже существует, его контент будет перезаписан при открытии в этом режиме.

Следовательно, при каждом запуске программы весь старый контент будет удален, и в файле останутся только товары, найденные по заданному критерию на текущую итерацию.

Fig. 3. Output Example.

Output: ... и результат всё равно выдаётся неправильно, то посмотрите vlikely be that there might be a misunderstanding of whitespace in the 'ingredients' function.

Try adding the spaces using string formatting to ensure the exact formatting as specified in the task. If the issue persists, consider re-checking the task requirements and the implementation details.

Fig. 4. Wrong Language Output Example.

Output:
Например, можно добавить дополнительный параметр "indent"(по умолчанию 4) и использовать его для форматирования вывода:
Код студента

```
def ingredients(food, indent=4):  
    for item in food:  
        print(' ' * indent + item)
```

Fig. 5. Full Python Code Output Example.

The responses containing only one or two words in unexpected foreign languages were not heavily penalized. However, outputs that included substantial portions of text in a foreign language were penalized more heavily according to our criteria. Similarly, outputs containing complete Python code, despite explicit instructions prohibiting full solutions, were subject to significant penalties.

2.4 Decision Making

More than 20 medium-sized language models were considered during the pre-selection process. Since the task involves code, it was first suggested to consider “code” versions of common models: DeepSeek Code, CodeLlama and others. It was expected that pre-training on code-based tasks would allow the assistant to better understand the task condition and to assist the student more flexibly. In practice, it turned out that code models of small size (up to 10 billion parameters), which can be deployed locally, were not at all suitable for our task.

Their understanding of the code was sufficient to solve the question at hand, but they themselves lacked dialog “flexibility”. They do not follow instructions well, they have narrower Russian language support and as a side effect of additional training on code tasks – they almost always give a full-fledged code or problem solution in the answer, which is inadmissible by the subject area of the task. Having found such disadvantages in the dialog part, we decided to turn to more general Instruction models, which are immediately designed for dialog with the user. At the same time, experiments showed that even such models already have enough knowledge about programming to fully help students solve problems from the course.

The initial selection resulted in the following set of alternatives:

- A1. Llama 3 8B IT [6];

- A2. Gemma 2 9B IT [7];
- A3. Qwen2.5 7B IT [8].

They have similar weights and also have moderate requirements for computational resources. Next, it was necessary to prioritize them according to the given criteria, and select the most suitable one. For this purpose, the classical TOPSIS [9] approach was used. First, we formed approximate preliminary estimates (Table 2).

Table 2. Preliminary Scores.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
A1	174	68	81	4	0	[7,8]	4	[7,8]	[7,9]
A2	160	94	97	0	0	[8,9]	3	[8,9]	[9,10]
A3	204	72	93	0	0	[6,7]	5	[6,7]	[9,10]

Since the solution has to be evaluated comprehensively and the criteria are of different nature, defuzzification of the scores by taking the mean value of the interval was chosen (Table 3).

Table 3. Defuzzified Scores.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
A1	174	68	81	4	0	7.5	4	7.5	8
A2	160	94	97	0	0	8.5	3	8.5	9.5
A3	204	72	93	0	0	6.5	5	6.5	9.5

Let's perform normalization of the estimates. For each column we calculated square roots from the sum of squares of its values, and then each column value was divided by the calculated root (Table 4). Next, weights were applied to the estimates (Table 5).

Table 4. Normalized Scores.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
A1	0.566	0.481	0.498	1.000	0	0.565	0.566	0.565	0.541
A2	0.520	0.665	0.597	0.000	0	0.640	0.424	0.640	0.642
A3	0.663	0.509	0.572	0.000	0	0.489	0.707	0.489	0.642

Table 5. Weighted Normalized Scores.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
A1	0.0283	0.0721	0.0498	0.1	0	0.0282	0.0566	0.1412	0.0811
A2	0.0260	0.0998	0.0597	0	0	0.0320	0.0424	0.1600	0.0963
A3	0.0332	0.0764	0.0572	0	0	0.0244	0.0707	0.1222	0.0963

The next step is to select Positive Ideal Solution (PIS) and Negative Ideal Solution (NIS). The PIS vector will contain maximum values for benefit criteria and minimum values for non-benefit criteria, and opposite for NIS (Table 6).

Table 6. Positive/Negative Ideal Solutions.

	C1	C2	C3	C4	C5	C6	C7	C8	C9
PIS	0.0332	0.0998	0.0597	0	0	0.0244	0.0707	0.1600	0.0963
NIS	0.0260	0.0721	0.0498	0.1	0	0.0320	0.0424	0.1222	0.0811

Next, based on the PIS and NIS values for the scores of each alternative, the Euclidean distances to each alternative were calculated. The final weight for each alternative was the ratio of the distance to the negative solution to the sum of the distances (Table 7).

Table 7. Results of Ranking.

	d_P	d_N	$\frac{d_N}{d_P+d_N}$	Final Rank
A1 (Llama 3 8B IT)	0.1081	0.0241	0.1823	3
A2 (Gemma 2 9B IT)	0.0302	0.1119	0.7873	1
A3 (Qwen2.5 7B IT)	0.0445	0.1059	0.7041	2

According to the obtained scores, Gemma 2 received the highest rank – this makes it the most preferable alternative in our decision-making task. Compared to its analogs, it demonstrated better understanding of the Russian language, generated correct hints and clearly followed the instructions.

3. Emotional State Awareness

One of the key aspects in the design of the assistant was to ensure not only methodologically correct but also psychologically comfortable interaction with students. While earlier sections focused on the relevance and contextual adaptation of the assistant's prompts, the current section addresses the integration of emotional state recognition as a critical feature, especially in remote learning environments where immediate human feedback is absent.

Emotional well-being directly affects learning efficiency, particularly in programming education, where students often encounter repeated failures, misunderstandings, or anxiety about "not knowing enough." These factors can lead to decreased motivation and even burnout. Therefore, lightweight but effective mechanism for emotional support was implemented, enabling the assistant to adjust its communicative tone based on the learner's psychological state.

The assistant evaluates the emotional state of the user on a scale from 1 to 10, where:

- 1-3 indicates high stress,
- 4-6 corresponds to neutral or mildly anxious states,
- 7-10 implies calmness, confidence, and concentration.

In accordance with the assistant's adaptive response mechanism, its tone and level of support are modulated based on the emotional state inferred from the user's input. The following examples, generated using the Gemma 2 model, demonstrate this adaptive behavior within the context of a typical Python programming task, specifically involving CSV file processing. Notably, the assistant provides distinct responses to the same user query, contingent upon the emotional context identified by the model. For more detailed information about the full prompt and its application, please refer to the Fig. 6.

Case 1: Emotional State = 6 (neutral) – *"Привет! Ошибка может возникнуть, если в некоторых строках CSV-файла меньше трех элементов. В этом случае row[2] вызовет IndexError: list index out of range, поскольку индекс 2 выходит за пределы списка".*

The assistant uses a factual and concise tone, focusing on the problem with minimal affective language.

Case 2: Emotional State = 2 (stress) – *"Привет! Понимаю, это может быть стрессово — с этим сталкивается каждый время от времени. Ошибка, вероятно, возникает, если в строке файла goods.csv содержится меньше трех элементов. Попытка обратиться к row[2] в таком случае вызывает IndexError. Не переживай — это очень распространенная проблема при работе с данными в формате CSV".*

Here, the assistant provides emotional reassurance alongside technical guidance, helping to mitigate frustration and support emotional resilience.

Case 3: Emotional State = 10 (confident) – *"Эта ошибка возникает, если в некоторых строках файла goods.csv меньше трех столбцов. Попытка доступа к row[2] в таких случаях вызывает ошибку IndexError. Рекомендуется добавить проверку длины перед доступом к индексу 2".*

The tone is business-like and information-dense, assuming the student requires direct input without emotional cushioning.

Rationale and Impact

Learning is inherently an emotional process, with cognitive effectiveness strongly influenced by the student's affective state. The assistant's ability to adapt its tone, even based on a coarse-grained numeric estimate, improves communication quality and reduces the risk of emotional burnout. This adaptive behavior aligns with the assistant's role as not only a technical but also a pedagogical tool, bridging the gap between purely algorithmic feedback and human-centered educational support.

Привет! Ты выполняешь роль ассистента по курсу программирования на Python. Перед тем как отвечать на вопрос студента, пожалуйста:

- Оцени по вопросу студента его эмоциональное состояние по шкале от 1 до 10, где:

- 1 - очень напряжен/раздражён,
- 10 - чувствую себя уверенно и спокойно.

- В зависимости от оценки адаптируй свой тон и подачу ответа:

1-3: Будь особенно поддерживающим. Подбодри студента, вырази понимание, говори мягко, избегай критики. Напоминай, что ошибки - это нормально и часть обучения.

4-6: Сохраняй дружелюбный и нейтральный тон. Дай чёткий ответ, избегая излишней строгости, подбодри при необходимости.

7-10: Используй деловой, уверенный и по существу стиль. Минимум эмоционального окраса, максимум пользы и точности.

Теперь сама задача:

Задача: Напишите программу, которая выбирает из таблицы товары, не дороже заданной цены.

Формат ввода: Файл goods.csv (UTF-8, разделитель - ;) в последней колонке записана цена товара. Целое число - верхняя граница цены - считывается с клавиатуры.

Формат вывода: Файл filtered-good.csv, содержащий товары с ценой не выше заданной, в исходном порядке.

Код студента:

```
import csv

def filter_goods(filename, max_price):
    with open(filename, encoding='utf-8') as file:
        reader = csv.reader(file, delimiter=';')

        with open('filtered-goods.csv', 'w', encoding='utf-8', newline='') as output:
            writer = csv.writer(output, delimiter=';')

            for row in reader:
                if int(row[2]) <= max_price:
                    writer.writerow(row)
```

Вопрос студента:

Почему этот код может вызвать ошибку при выполнении? В каком месте возможна проблема?

Ограничения на ответ:

В ответе не должно быть исправленного кода.

Пример пояснения ошибки:

Ошибка может возникнуть из-за того, что студент принимает row[2] за цену товара, не проверив, что строка действительно содержит три значения. Это может привести к Index Error: list index out of range.

Проблема возможна в строке if int (row [2]) <= max_price:

Fig. 6. Mistakes Handling Example.

4. Aspects of Software Implementation

4.1 System Architecture

The software implementation of the "Sage" project comprises several key components (Fig. 7) that interact to deliver adaptive hints to students within the Moodle learning platform. The system is designed using a microservice architecture to ensure flexibility, scalability, and responsibility segregation between modules. It consists of the following important components:

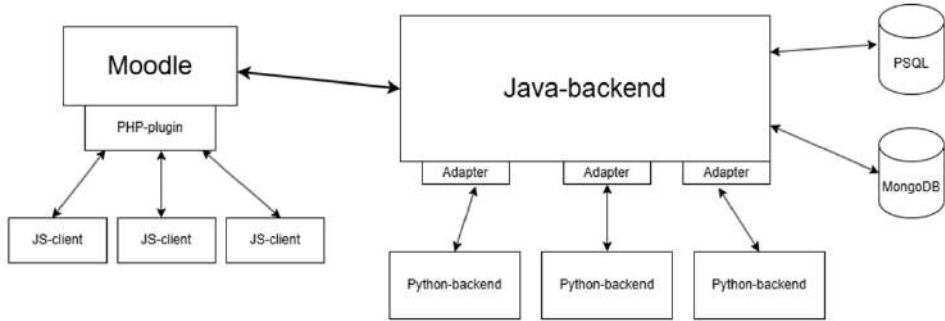


Fig. 7. Overall System Architecture.

- **Client-side (JS Client):** Implemented in JavaScript and integrated into Moodle as part of a PHP plugin. It is responsible for the user interface (UI) components such as the chat window and code highlighting functionality, real-time user interaction, context gathering (e.g., task details), and dispatching requests to the server. The client architecture features a universal API for chat interaction and context transmission, facilitating straightforward functional extensions.
- **Moodle Plugin (PHP Plugin):** Operates within the Moodle environment. It functions as a bridge between the JS client and the main backend logic. Leveraging Moodle's modular plugin system, it provides access to course and user data, validates and complements client requests with the required context (e.g., assignment information) before forwarding them to the main server. It is implemented as a block plugin.
- **Main Server (Java Backend):** Acts as the central mediator within the microservice architecture. It receives context-complemented requests from the PHP plugin, manages business logic, interacts with the databases, and coordinates requests to the AI module via adapters. The Java backend architecture is designed with configurability in mind.
- **AI Module (Python Backend):** A microservice responsible for processing requests using a Large Language Model (LLM). It receives requests from the Java backend through adapters, interacts with the LLM to generate explanations, hints, and code analysis, and returns the responses. Key functions include status monitoring and processing textual requests, taking into account context and system instructions.
- **Databases:** The system utilizes two Database Management Systems (DBMS): a relational one (PostgreSQL - PSQL) and a non-relational one (MongoDB). This separation allows storing structured data (e.g., chat history) in PostgreSQL, and more flexible or large-scale data (e.g., complex contexts in JSON format) in MongoDB.
- **Adapters:** Components ensuring standardized interaction between the Java backend and various Python backend implementations (AI modules).
- **Deployment:** The entire system is packaged using Docker and Docker Compose, ensuring ease of deployment on any host, managing dependencies via a shared network, and enabling easy data resets (purging) for testing or updates. For code execution in an isolated environment (if required), Jobe can be utilized.

4.2 Client Application (JS Client)

The client application, written in JavaScript, serves as the primary interface for user interaction with the "Sage" system within Moodle. Main features of this application are following:

- **UI Components:** Features a chat window and buttons enabling users to request explanations for selected code snippets or analyses of runtime errors.
- **Interaction:** Utilizes asynchronous requests (Ajax) directed to the PHP plugin for sending messages and receiving responses. It also processes server responses, displaying them to the user.
- **Context:** Transmits not only the user's message but also relevant context, such as the selected code fragment or error text.
- **Extensibility:** The API is designed to easily accommodate new request types (e.g., requests for motivational messages or specific types of hints).

4.3 Server Components (PHP & Java)

The server-side infrastructure is divided into two main components: the PHP plugin for Moodle integration and the Java backend acting as a mediator:

- **PHP Plugin:** Developed adhering to Moodle's API and architectural standards. It receives data from the JS client, verifies access permissions and request validity, extracts necessary context (User ID, Course ID, Task/Assignment ID, etc.) from the Moodle environment, complements the original request with this context, and securely forwards it to the Java backend.
- **Java Backend:** Implemented using Java (leveraging the Spring Framework), it acts as the central hub: routing requests, managing chat sessions, persisting request history to MongoDB, querying PostgreSQL for chat content, and interacting with the Python backend via a standardized interface (adapters). The data partitioning between PostgreSQL and MongoDB is chosen for optimal storage of different data types.

5. Conclusion

The evaluation results revealed that Gemma 2 9B IT emerged as the most suitable model, offering the best overall alignment with the proposed criteria, such as accurate instruction following, context comprehension, and effective hint generation in Russian language.

To fully realize the capabilities of the assistant, the necessary software infrastructure was developed. This includes a universal PHP plugin for seamless integration into Moodle courses, a modular backend (Java and Python) for request routing and AI processing, and a flexible adapter system to simplify the future addition of alternative LLMs.

The resulting system represents a scalable and adaptable educational tool, capable of being extended to support other courses or reconfigured in response to changes within the current course structure. Its modular architecture ensures both ease of maintenance and extensibility.

Future work will focus on expanding the affective and psychological dimension of the assistant. In particular, upcoming versions will incorporate mechanisms for long-term emotional profiling, taking into account not only the student's current affective state but also their interaction history. This will allow the assistant to offer increasingly personalized and empathetic support adapted to the individual students.

References

- [1]. M. A. Razafinirina, W. G. Dimbisoa, and T. Mahatody, "Pedagogical alignment of large language models (llm) for personalized learning: a survey, trends and challenges," *Journal of Intelligent Learning Systems and Applications*, vol. 16, pp. 448–480, 2024.

- [2]. R. E. Wang, A. T. Ribeiro, C. D. Robinson, S. Loeb, and D. Demszky, "Tutor copilot: a human-ai approach for scaling real-time expertise," arXiv preprint arXiv:2410.03017, 2024.
- [3]. T. Phung et al., "Automating human tutor-style programming feedback: leveraging gpt-4 tutor model for hint generation and gpt-3.5 student model for hint validation," in Proceedings of the 14th Learning Analytics and Knowledge Conference, 2024, pp. 12–23.
- [4]. L. Roest, H. Keuning, and J. Jeuring, "Next-step hint generation for introductory programming using large language models," in ACE '24: Proceedings of the 26th Australasian Computing Education Conference, 2024, pp. 144–153.
- [5]. Data for article: prompts and answers. (Online). Доступно по ссылке: https://github.com/sage-hse/data_for_article
- [6]. M. AI, "Llama 3 technical report," <https://ai.meta.com/llama/>, 2024, accessed: 2025-04-10.
- [7]. M. Riviere, S. Pathak, P. G. Sessa, and et al., "Gemma 2: Improving open language models at a practical size," arXiv preprint arXiv:2408.00118, 2024. (Online). Доступно по ссылке: <https://arxiv.org/abs/2408.00118>
- [8]. A. Yang, B. Yang, B. Cao, and et al., "Qwen2: Scaling up and distilling down open multilingual language models," arXiv preprint arXiv:2403.16473, 2024. (Online). Доступно по ссылке: <https://arxiv.org/abs/2403.16473>
- [9]. E. Roszkowska, "Multi-criteria decision-making models by applying the topsis method to crisp and interval data," in Multiple Criteria Decision Making. Springer International Publishing, 2011, pp. 7–11.

Information about authors

Екатерина Андреевна КАРАВАЕВА является стажером-исследователем лаборатории облачных и мобильных технологий факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики». Сфера её научных интересов охватывает облачные технологии, алгоритмы и структуры данных.

Ekaterina Andreevna KARAVAeva is a research intern at the Laboratory of Cloud and Mobile Technologies, Faculty of Computer Science, National Research University Higher School of Economics. Her research interests include cloud technologies, algorithms, and data structures.

Владимир Игоревич ВАСИЛЕВСКИЙ является стажером-исследователем лаборатории облачных и мобильных технологий факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики». Сфера его научных интересов включает облачные технологии и кодогенерация.

Vladimir Igorevich VASILEVSKIJ is a research intern at the Laboratory of Cloud and Mobile Technologies, Faculty of Computer Science, National Research University Higher School of Economics. His research interests include cloud technologies and code generation.

Георгий Михайлович ЛАНИН является студентом 2-го курса программы системная и программная инженерия департамента программной инженерии факультета компьютерных наук Национального исследовательского университета «Высшая школа экономики». Сфера его научных интересов охватывает системный анализ и мультиагентные системы.

Georgy Mikhailovich LANIN is a 2nd year student of the System and Software Engineering program at the Department of Software Engineering, Faculty of Computer Science, National Research University Higher School of Economics. His research interests include systems analysis and multi-agent systems.

Дмитрий Сергеевич ПРОКУДИН является специалистом кафедры Математических методов прогнозирования Московского государственного университета имени М.В. Ломоносова. Его научные интересы включают распознавание образов.

Dmitrii Sergeevich PROKUDIN – specialist of the Department of mathematical methods of forecasting of CMC of Lomonosov Moscow State University. His research interests include pattern recognition.

DOI: 10.15514/ISPRAS-2025-37(4)-26



Integrating an Ontology-Driven Approach to Data Visualization and AI-Based Visualization with Plotly

A.D. Dzheiranian, ORCID: 0009-0000-8916-2855 <ADDzheyryanyan@edu.hse.ru>

L.N. Lyadova, ORCID: 0000-0001-5643-747X <LNLyadova@gmail.com>

HSE University,

38, Studencheskaia St., Perm, 614070, Russia

Abstract. This study introduces an AI-driven assistant prototype that automates the generation of data visualization scripts from natural language queries, eliminating the need for users to have programming skills. The article examines research aimed at developing tools for effective data visualization, compares data visualization systems based on the use of artificial intelligence, and shows the limitations of the existing tools. The proposed approach to data visualization is based on integrating knowledge-driven DSM platform (language toolkits) and generative AI tools. The proposed methodology categorizes tasks of data visualization into two distinct types: standard and non-standard. Standard tasks are solved with a code-generation approach based on prompts within a visual environment. Non-standard tasks are handled by extending existing libraries with user-defined packages. The language-oriented approach with DSM tools effectively unifies both categories: for standard tasks, users work with pre-existing DSLs and adjust parameters as necessary, whereas for non-standard tasks, users develop new DSLs with language toolkits automating visual DSL creation and code generation. The core of the language toolkits is multifaceted ontology. By integrating a large language model (LLM) with a knowledge-driven framework and a multifaceted ontology, the system enables dynamic, context-aware visualization workflows that ensure semantic traceability and reproducibility. The ontology not only stores descriptions of data visualization tasks but also facilitates the reuse of generated scripts, thereby enhancing the system's adaptability and fostering collaborative analytical work among user communities. The dataset, containing entries and variables encompassing different domains, is used to demonstrate the functionality of the prototype. The article provides examples of developing several visualization options, demonstrating the application of the proposed approach. Case studies demonstrate the prototype's efficacy in creating histograms, scatter plots, and other visualization methods, while reducing technical barriers for users. Future work will extend the assistant's functionality by incorporating user-defined visualization packages and additional LLM training to address non-standard tasks and complex visualization scenarios.

Keywords: data visualization; artificial intelligence; domain specific modeling; language toolkits; ontology; Python; Dash; Plotly.

For citation: Dzheiranian A.D., Lyadova L.N. Integrating an Ontology-Driven Approach to Data Visualization and AI-Based Visualization with Plotly. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025, pp. 191–206. DOI: 10.15514/ISPRAS-2025-37(4)-26.

Интеграция подхода к визуализации данных на основе онтологии и визуализации на основе ИИ с использованием Plotly

А.Д. Джейранян, ORCID: 0009-0000-8916-2855 <ADDzheyranyan@edu.hse.ru>

Л.Н. Лядова, ORCID: 0000-0001-5643-747X <LNLyadova@gmail.com>

Национальный исследовательский университет «Высшая школа экономики»,
Россия, 614070, г. Пермь, ул. Студенческая, д. 38.

Аннотация. В данном исследовании представлен прототип ассистента, управляемого искусственным интеллектом, который автоматизирует генерацию скриптов визуализации данных на основе запросов на естественном языке, устраняя необходимость пользователей владеть навыками программирования. В статье рассматриваются исследования, направленные на разработку средств для эффективной визуализации данных, проводится сравнение систем визуализации данных, основанных на использовании искусственного интеллекта, показаны ограничения существующих средств. Предлагаемый подход к визуализации данных основан на интеграции DSM-платформы (языкового инструментария), управляемого знаниями, и инструментов генеративного искусственного интеллекта. Предлагаемая методология разделяет задачи визуализации данных на два типа: стандартные и нестандартные. Стандартные задачи решаются с помощью генерации кода на основе подсказок в визуальной среде. Нестандартные задачи обрабатываются путем расширения существующих библиотек с помощью определяемых пользователем пакетов. Языково-ориентированный подход с использованием средств DSM эффективно объединяет обе категории: для стандартных задач пользователи работают с существующими DSL и настраивают параметры по мере необходимости, тогда как для нестандартных задач пользователи разрабатывают новые DSL с помощью языкового инструментария, автоматизирующего создание визуальных DSL и генерацию кода. Ядром языкового инструментария является многоаспектная онтология. Благодаря интеграции большой языковой модели (LLM) с фреймворком, управляемым знаниями, и многоаспектной онтологией, система обеспечивает динамические, контекстно-ориентированные рабочие процессы визуализации, которые гарантируют семантическую прослеживаемость и воспроизводимость. Онтология не только хранит описания задач визуализации данных, но и облегчает повторное использование сгенерированных скриптов, повышая тем самым адаптивность системы и способствуя совместной аналитической работе сообществ пользователей. Набор данных, содержащий записи и переменные, охватывающие различные предметные области, используется для демонстрации функциональности прототипа. В статье приведены примеры разработки нескольких вариантов визуализаций, демонстрирующие применение предлагаемого подхода. Приведённые примеры демонстрируют эффективность прототипа в создании гистограмм, точечных диаграмм и других методов визуализации и снижении технических барьеров для пользователей. В перспективе функциональность ассистента будет расширена за счёт поддержки пользовательских пакетов визуализации и дополнительного обучения LLM для решения нестандартных задач и сложных сценариев визуализации.

Ключевые слова: визуализация данных; искусственный интеллект; предметно-ориентированное моделирование; языковой инструментарий; онтология; Python; Dash; Plotly.

Для цитирования: Джейранян А.Д., Лядова Л.Н. Интеграция подхода к визуализации данных на основе онтологии и визуализации на основе ИИ с использованием Plotly. Труды ИСП РАН, том 37, вып. 4, 2025 г., стр. 191–206 (на английском языке). DOI: 10.15514/ISPRAS-2025-37(4)-26.

1. Introduction

The field of data visualization has gained significant academic attention, as evidenced by a 77% increase in related publications in the Scopus database, rising from 3 232 in 2015 to 5 724 in 2024. In data-driven decision-making, visualization tools play a crucial role in interpreting complex datasets, identifying patterns, and supporting insightful decision processes. These tools are used by a wide range of professionals, including data analysts, scientists, business executives, managers, educators, students, and other stakeholders.

Despite the growing reliance on data visualization, many users encounter substantial challenges in generating effective and accurate visual representations. A major concern is the proliferation of low-quality or misleading visualizations that either fail to convey meaningful insights or even distort the underlying data [1]. This issue is widely discussed in communities such as “Data is Ugly” (www.reddit.com/r/dataisugly/), where enthusiasts and experts alike showcase and critique poor visualization practices. To address these challenges, the integration of user-defined visualization specifications is essential [2]. These specifications enable users to customize visualizations according to the specific requirements of their tasks and domains, ensuring a more effective and informative presentation of data.

Current solutions – including spreadsheets, business intelligence (BI) platforms, programming languages with libraries for data visualization, domain specific languages (DSLs) for customizing visualization models, and approaches leveraging LLMs – offer varied levels of customization and implementation of specifications. However, in most cases, customization is limited to basic parameter adjustments, which may not always be sufficient. Some scenarios require more advanced modifications, such as the ability to create custom chart types. For example, designing a basketball specific scatter plot, where the plotting area mimics the layout of a basketball court rather than a basic rectangle, etc. [3].

Many researchers also highlight the inherent complexity of creating high-quality visualizations. As noted in [4], traditional visualization methods often require high programming expertise for customizing models, which poses significant challenges for users. Similarly, [2] emphasizes that designing effective visualizations remains a time-consuming and complex task, even for experienced professionals.

Thus, there is a growing need for a novel approach that enables the customization of diagrams to align with domain specific requirements and task objectives – without the necessity for advanced programming skills.

In previous studies, a promising data visualization methodology that categorizes visualization tasks into standard and non-standard ones was proposed. The platform architecture adheres to knowledge-driven principles, with a multifaceted ontology serving as the central repository for expert knowledge [5-7]. One method for addressing data visualization tasks within this methodology involves the generation of scripts in a programming language, thereby simplifying the workflow for non-technical users when developing and customizing diagrams.

Building upon this foundation, the present research aims to design and implement a prototype AI-assistant for the automated generation of visualization scripts, integrated with knowledge-driven architecture. The tasks of the study include:

- 1) an analysis of existing tools for automated visualization generation;
- 2) the architectural design of the AI-assistant prototype;
- 3) the prototype implementation, including the development of the subontology;
- 4) the demonstration of the prototype’s functionality using a representative dataset.

2. Related Works

Programming languages offer extensive capabilities for creating effective and customizable visual representations. The most widely used languages in this field are Python, R, and JavaScript, each providing a rich ecosystem of visualization libraries [8]. Maximum data visualization customization is provided for users, but coding skills are required. Specialized libraries reduce this barrier, though a basic understanding of programming remains essential. Recent advances in artificial intelligence have begun bridging this gap by automating data visualization workflows [9-11], particularly through natural language interfaces and script generation.

Modern AI-driven systems leverage large language models like GPT-3 and ChatGPT to interpret user queries, generate visualization scripts, and refine outputs iteratively. For instance, Chat2VIS

demonstrates how prompt engineering can produce context-aware visualizations while maintaining data security [12]. Similarly, the platform described in [4] integrates NLP techniques and generative AI with Pandas to automate Python script generation for CSV-based datasets. ChatVis [13] is an iterative assistant built atop GPT-4 that synthesizes Python scripts for scientific visualization. Using an error-feedback loop, ChatVis captures runtime exceptions from each script execution and resubmits them to the model for correction, repeating until the code runs successfully. Data Formulator 2 [14] combines a drag-and-drop UI with natural-language instructions to define new data fields and desired charts. It compiles UI selections, dialogue context, and NL directives into an LLM prompt, then generates Python code for both data transformation and visualization in an interactive, iterative session.

However, current AI-driven visualization systems exhibit critical limitations (Table 1).

Table 1. Limitations of current AI-driven visualization systems

Limitation	Description
Model “hallucinations”	LLMs may produce syntactically invalid code, or misinterpret parameters, undermining trust and necessitating expert oversight of generated scripts [12, 13]
Poor handling of ambiguity	When faced with vague or multi-interpretation queries, systems often generate overly generic or incomplete visualizations, failing to capture the full nuance of user intent in complex analytical scenarios [12]
Restricted chart types	The use of standard libraries (for example, Matplotlib, Pandas) prevents the creation of non-standard visual forms, which forces users to refine low-level code manually
Limited creative design support	Customization options rarely go beyond chart type, color palette, and basic annotations; detailed control over layout, styling, and individual visual elements remain largely unsupported
Lack of domain specific expertise	Open models trained on broad web data struggle with specialized terminology and industry standards, leading to misinterpretation of subject-area concepts [15]

While current LLMs cannot fully replace human expertise and still require robust verification and fine-tuning frameworks, they hold significant potential for accelerating code generation in data-analysis and visualization workflows.

This research develops and integrates a natural language script generation method into a broader knowledge-driven data visualization platform. The AI-assistant allows users to describe their charting requirements in natural language, automatically generates and runs the corresponding Python scripts, and uses a multifaceted ontology to audit user queries and facilitate the reuse of generated code across similar scenarios. The platform’s library extension subsystem is designed to further train the model on custom visualization packages, empowering it to tackle domain specific, non-standard plotting tasks.

3. Data Visualization Technique Based on Knowledge and Generative AI

3.1 Proposed Approach to Data Visualization

The proposed methodology categorizes tasks of data visualization into two distinct types: standard and non-standard. For *standard tasks*, a code-generation approach based on prompts within a visual environment is employed, which facilitates the automatic creation and execution of scripts. In contrast, *non-standard tasks* are handled by extending existing libraries with user-defined packages. The *language-oriented approach* effectively unifies both categories: for standard tasks, users work with pre-existing DSLs and adjust parameters as necessary, whereas for non-standard tasks, users develop new DSLs. This development process is streamlined through the automated mapping of the domain ontology onto the metamodel of a base language, as described in [5-7].

The system architecture design follows a *knowledge-driven approach*, where the core of the platform is a *multifaceted ontology*. This ontology stores essential knowledge across the platform and is organized into six primary groups:

1. *Data sources ontology* includes information on the structure, types of data, etc.
2. *Domain ontology* contains the subjective knowledge of the expert community: concepts of the subject area, their relationships, and limitations (including rules).
3. *DSM-knowledge ontology* encompasses models, visualization languages, subject domains, rules for the transformation and generation of DSLs, etc.
4. *Scripts ontology* captures information regarding prompts, generation outcomes, and relevant metadata.
5. *Packages ontology* contains detailed descriptions of user-defined visualization packages.
6. *User task repository* is an ontology of user tasks and applied methods. Using this ontology facilitates search and reusable deployment of established models.

The system's overall architecture is shown in Fig. 1.

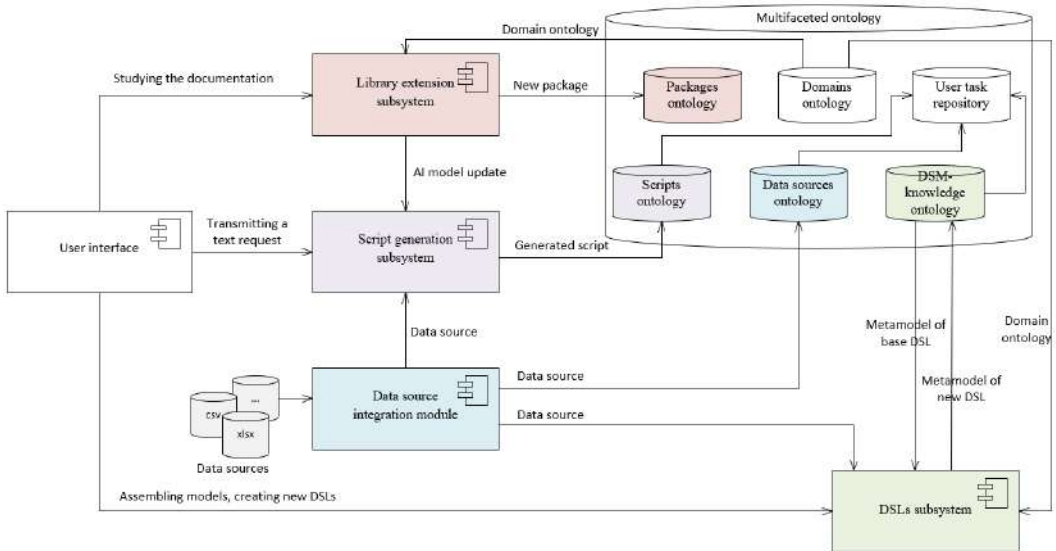


Fig. 1. Generalized structure of the data visualization tools prototype

In addition to the ontology, the system consists of the following components:

1. *User interface* is implemented as a single-entry point: the UI unifies three isolated, specialized interfaces (visualization library management, script generation, and DSM-platform interaction) behind a Facade pattern that provides seamless navigation between sections without exposing their internal logic.
2. *Library extension subsystem* is responsible for creating new packages atop existing visualization libraries (e.g., Matplotlib, Seaborn, D3, etc.) and for training the language model to understand and leverage these extended libraries.
3. *Script generation subsystem* automatically generates and executes visualization scripts based on user input.
4. *DSLs subsystem* constitutes the DSM-platform (language toolkit), manages both the utilization of preexisting DSLs and the creation of new ones through an automated mapping of the domain ontology onto the metamodel of a base language.

5. *Data source integration module* provides connections to CSV and XLSX data stores. It uses the *Data sources ontology* to automatically save metadata about any dataset uploaded to the system.

3.2 AI-assistant Architectural Layers and Component Responsibilities

The AI-assistant is presented as a *Script generation subsystem*, that integrates semantic technologies with LLMs to automate visualization code generation, while maintaining an ontology-grounded history of user interactions. The system adheres to layered architecture (Fig. 2), decoupling *presentation*, *business logic*, and *data management* concerns to ensure extensibility and maintainability.

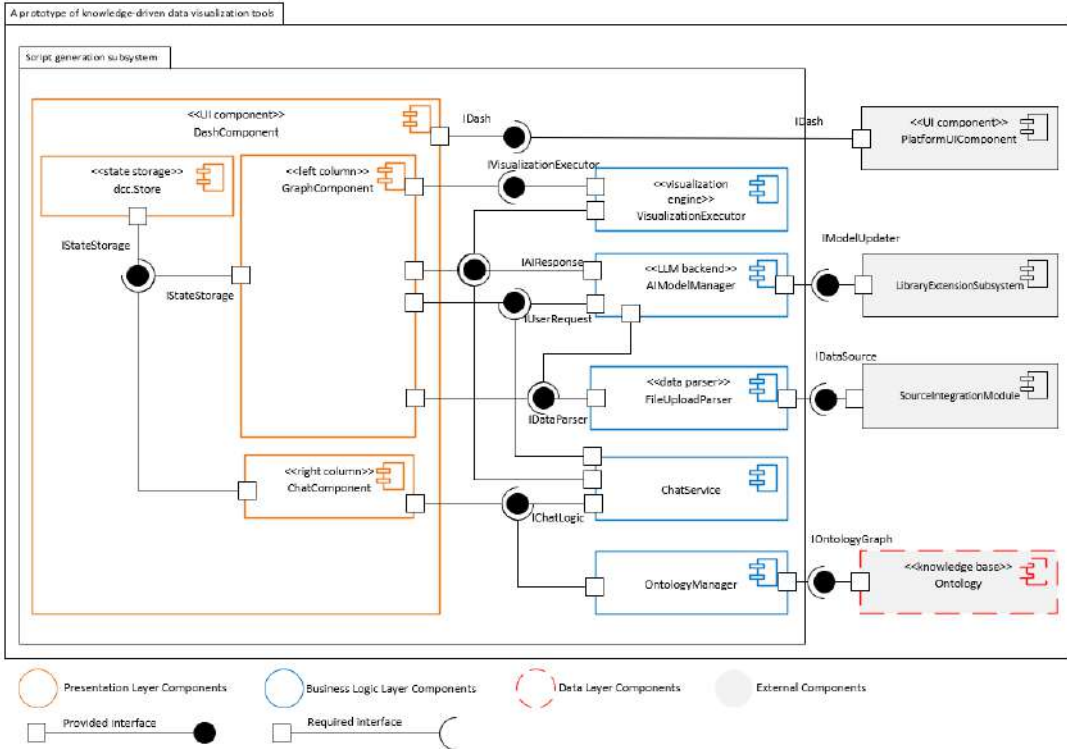


Fig. 2. *Script generation subsystem (AI-assistant) architecture*

The *presentation layer* follows a reactive programming model, with Dash callbacks mediating between user actions and backend services. The *DashComponent* serves as the root container, implementing the user interface via the Dash framework. It orchestrates three subcomponents:

1. *ChatComponent* manages real-time chat interactions through the *IChatLogic* interface, rendering message history and propagating user inputs.
2. *GraphComponent* handles visualization outputs by coordinating dependencies through three critical interfaces: *IDataParser* – for structured data ingestion; *AIResponse* – for displaying the generated LLM response; *IVisualizationExecutor* – for dynamic code execution. It also implements the *IUserRequest* interface to expose user-entered queries.
3. *dcc.Store* acts as an in-browser, client-side state container. It provides the *IStateStorage* interface for any UI component to read or write transient JSON data (e.g., current query, graph parameters, or intermediate results). This component eliminates unnecessary server roundtrips by keeping frequently accessed state in the browser.

Five components of the *business logic layer* implement domain specific processing:

1. *FileUploadParser (IDataParser)* implements CSV/Excel parsing via Pandas, extracting columnar data.
2. *ChatService (IChatLogic)* manages chat session state, enforcing conversation context persistence.
3. *AIModelManager (IAIResponse)* utilizes LangChain pipelines with the ChatGroq API (Llama3-70B) to generate Plotly visualization code, conditioned by uploaded data (*IDataParser*) and user request (*IUserRequest*).
4. *VisualizationExecutor (IVisualizationExecutor)* safely executes generated Python code in sandboxed environments.
5. *OntologyManager* maintains the RDFlib knowledge graph, implementing ontology operations (prompt/script entity creation, model versioning, etc.) through *IOntologyGraph*.

The *Ontology* component of the *data layer*, exposed through the *IOntologyGraph* interface, acts as the system's persistent knowledge base.

The *Script generation subsystem* interacts with *external components* (third-party prototype components):

1. The *PlatformUIComponent* is the core user interface of the platform. It integrates the *DashComponent* to display visualization and chat interactions via linked interface *IDash*.
2. The *LibraryExtensionSubsystem* is responsible for developing and publishing new visualization packages. It implements the *IModelUpdater* interface, through which the *AIModelManager* component initiates an update or selection of the LLM model version. When preparing a query to a language model, the *AIModelManager* switches the context to the latest trained model via *IModelUpdater* if necessary.
3. When a new file is uploaded to the system, the *SourceIntegrationModule* passes it to the *FileUploadParser* component via the *IDataSource* interface for further processing.

3.3 User-System Interaction Workflow

The *data visualization assistant* operates through a multistage process (Fig. 3), which is initiated once a user uploads a data file. This workflow can be summarized as follows:

1. *File upload and data parsing.* After the user uploads a data file (CSV or Excel), the system validates the file to confirm compatibility with supported data types. Then it converts the contents into a structured representation suitable for subsequent processing and displays an interactive preview, allowing users to inspect and verify the data before proceeding.
2. *User request processing.* When a visualization request is submitted, the application constructs a context-aware prompt to provide the system with sufficient details for generating suitable visualization. This information is passed on to the generative model, which produces an executable script tailored to the user's stated objective.
3. *Code execution and visualization.* The generated script undergoes a basic sanitization process to ensure compliance with the application's requirements. Subsequently, it is executed in a contained environment. The visualization output is then rendered within the application interface, enabling users to explore the data interactively. Errors or inconsistencies within the script are handled gracefully, with feedback provided to the user as needed.
4. *Semantic metadata persistence.* All pertinent data from each session (including user prompts, the resulting scripts, and associated metadata) are stored in a semantic repository.
5. *Chat history and interface updates.* The human–AI interactions are appended to a session-specific chat history, formatted with distinct styling for human–AI messages. The interface dynamically refreshes to display both the visualization and code snippet, ensuring transparency in the AI's output.

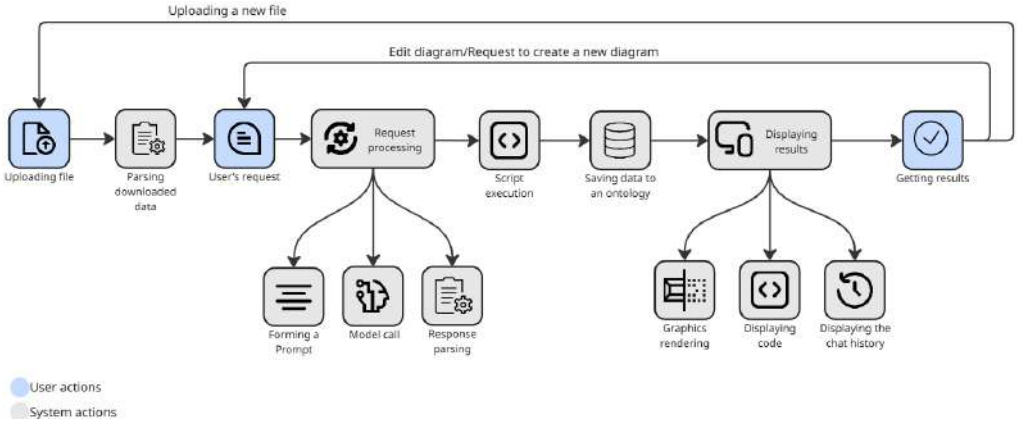


Fig. 3. User-system interaction workflow

This workflow emphasizes automation, reproducibility, and semantic traceability by integrating interactive visualization with knowledge graph technologies. The ontology facilitates long-term analytics on AI-generated artifacts, thereby adhering to FAIR (Findable, Accessible, Interoperable, and Reusable) data principles [16].

4. Prototype Development

4.1 Selection of Development Tools

Programming language Python with *Plotly* visualization library was selected for its versatility and extensive support across data analytics and visualization domains. *Plotly*, in particular, was chosen due to its capabilities for interactive visualization and its comprehensive suite of supported chart types.

The *ontological framework* was developed using *Protégé ontology editor*. This framework enables the persistence of necessary data within an RDF-based ontology managed via the *Python library RDFlib*.

The user interface is implemented using *Dash* – a Python framework designed for building interactive web applications.

AI integration for natural language processing and prompt management is implemented with the *Groq API* in conjunction with the *llama3-70b-8192 model*. This configuration provides an 8192-token context window, supporting extensive natural language input without limitations on maximum completion tokens or file size.

The system leverages the *LangChain* framework for the *pipeline orchestration* to orchestrate the end-to-end process of generating *Plotly-compatible Python code*. *LangChain* facilitates prompt formulation and task chaining. It ensures that the model adheres to predefined constraints (such as omitting data-loading steps) and focuses solely on visualization logic (e.g., axis mapping and chart type selection).

4.2 Ontology Development

The structure of the ontology comprises three core classes: *Prompt*, *GeneratedScript*, and *Metadata*, interconnected via object properties to model provenance relationships (Table 2). Data properties are defined to capture temporal, contextual, and technical attributes. The relationships are structured as follows: “*Prompt* → (*producedScript*) → *GeneratedScript* → (*hasMetadata*) → *Metadata*”. The ontology was serialized in RDF/XML to ensure interoperability with semantic web tools.

Table 2. Ontology entities

Class	Properties	
	Data properties	Object properties
Generated_scripts	hasCodeContent (string) hasLanguage (string) hasTimestamp (dateTime)	hasMetadata
Metadata	hasFileName (string) hasModelVersion (string)	—
Prompts	hasContent (string) hasSessionID (string) hasTimestamp (dateTime)	producedScript

4.3 Interface Design

The interface features a dual-panel layout (Fig. 4). Left Panel (*GraphComponent*) combines data upload functionality, visualization controls, and code display. Key components are:

- a drag-and-drop file uploader supporting CSV/Excel formats;
- an interactive grid (Dash AG Grid) for real-time data preview;
- a text area for natural language queries and a button to trigger AI processing;
- dynamic rendering of Plotly graphs and generated code snippets.

Right Panel (*ChatComponent*) displays a session-specific chat history with distinct styling for user inputs (blue background) and AI responses (grey background).

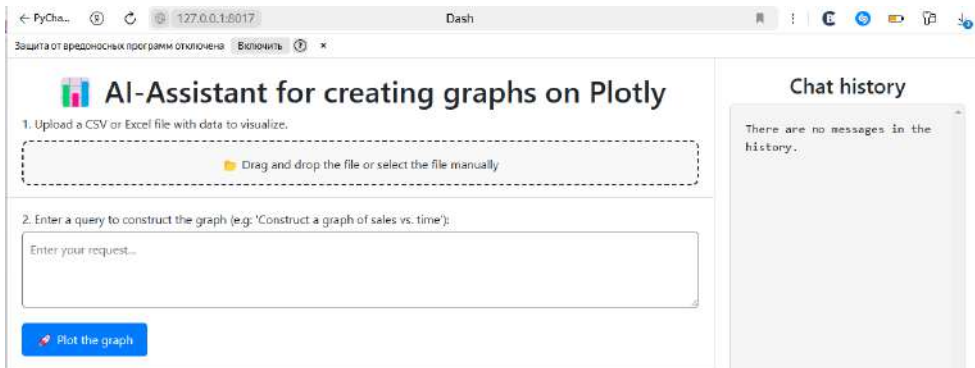


Fig. 4. AI-assistant interface

4.4 Prompt Formation

When a visualization request is submitted, the application constructs a context-aware prompt using LangChain's *ChatPromptTemplate*. The prompt integrates the following elements: the original filename; the first five rows of the dataset as a CSV string; historical chat messages (if available) to maintain conversational continuity; a user request.

After the prompt is generated, it is processed by the Groq API using the llama model.

4.5 Code Extraction and Visualization Execution

The *VisualizationExecutor* component orchestrates the transformation of AI-generated code into executable visualization workflows.

Regular expression patterns for the code transformation:

`r```(?:[Pp]ython)?(.*)```` (1)

`r'df\s*=\s*pd\.read_\w+(\..*?)'` (2)

`r'(?m)^\s*fig\.show\(\)\s*$'` (3)

Upon receiving the model’s raw output, a regular expression pattern (1) identifies and extracts Python code blocks. The sanitization phase removes non-essential operations, including: redundant data loading – predefined DataFrame declarations (`pd.read_csv()` / `pd.read_excel()`) are stripped (2), as the dataset is already cached client-side; rendering commands – calls to `fig.show()` are omitted (3) to prevent runtime conflicts within Dash’s callback architecture.

The purified code is executed dynamically via Python’s function `exec()` within a sandboxed environment.

4.6 Chat History Management

The *ChatService* component ensures conversational continuity by persisting user-AI interactions in a JSON-serializable format. Key features include the following functions:

- message typing: messages are classified as *HumanMessage* (user queries) or *AIMessage* (model outputs), each with timestamped metadata;
- state preservation: the *dcc.Store* component caches chat history client-side, enabling seamless navigation during active sessions;
- dynamic rendering: messages are displayed in chronological order with auto-scrolling to the latest entry.

4.7 Ontology-to-Prototype Integration

The integration of semantic knowledge representation with application logic is facilitated by a dedicated *OntologyManager* component. This component orchestrates the following processes:

1. Ontology Initialization:
 - the component loads an existing RDF/XML file or initializes a new RDF graph;
 - custom namespace (“*MY*”) is bound to ensure Uniform Resource Identifier (URI) consistency across entities, adhering to semantic web standards.
2. Individuals Generation:
 - the `add_prompt(content, session_id)` method generates unique prompt individuals, assigning Universally Unique Identifiers (UUIDs), user provided content, and optional session identifiers to enable cross-request traceability;
 - the `add_generated_script(code_content, language = “Python”)` function encapsulates generated code snippets, programming language specifications (default: Python), and timestamps to record the exact time of code synthesis;
 - the `add_metadata(file_name, model_version)` method associates scripts with technical metadata, including the source filename and AI model version, ensuring reproducibility.
3. Semantic Relationship Mapping:
 - RDF triples are dynamically constructed using object properties *producedScript* and *hasMetadata*;
 - the methods `link_prompt_to_script(prompt_uri, script_uri)` and `link_script_to_metadata(script_uri, metadata_uri)` formalize these connections within the ontology, establishing a provenance chain from user requests to technical execution parameters.

5. A Case of Data Visualization Generation

5.1 Dataset Description

To demonstrate the functionality of the prototype, the *Sleep_health_and_lifestyle_dataset.csv* was used. This dataset contains 400 entries and 13 variables, encompassing demographic, behavioral, and physiological factors related to sleep health, such as age, gender, occupation, sleep duration and quality, physical activity, stress, BMI category, blood pressure, heart rate, daily steps, and the presence of sleep disorders.

5.2 Experiments with Generating Data Visualizations

After uploading the dataset, the system displays the data in a tabular format, allowing users to filter and sort columns (Fig. 5). At this initial stage, the message history is empty.

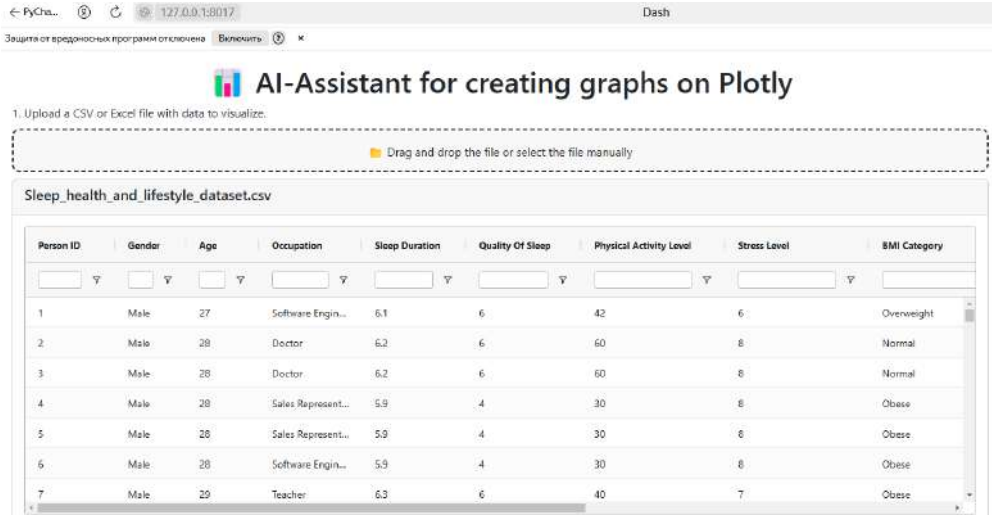


Fig. 5. Display of loaded data in the system

5.2.1 Step 1 “No Chart Type, Only Concept”

Task: “Build a visualization of the top 10 professions with the highest stress level”.

Response and commentary: the system autonomously selected an appropriate visualization type – a bar chart – to represent the data (Fig 6). The interactive graph displays tooltips upon hovering over a bar. Examination of the code (Fig. 7) reveals that the process involves grouping by occupation, computing the average stress level, and subsequently sorting the results to identify the top ten professions.

Top 10 Professions with the Highest Stress Level

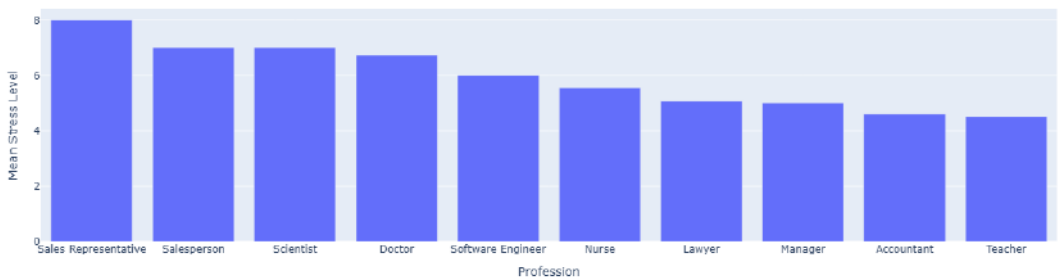


Fig. 6. Experiment №1 – Generated visualization

```
Here's the code to create a bar chart using Plotly to visualize the top 10 professions with the highest stress level:
... python
import pandas as pd
import plotly.express as px

# Load the data
df = pd.read_csv('Sleep_health_and_lifestyle_dataset.csv')

# Group by Occupation and calculate the mean Stress Level
stress_by_profession = df.groupby('Occupation')['Stress Level'].mean().reset_index()

# Sort the data by mean Stress Level in descending order
stress_by_profession = stress_by_profession.sort_values('Stress Level', ascending=False)

# Select the top 10 professions with the highest stress level
top_10_stressful_professions = stress_by_profession.head(10)

# Create a bar chart using Plotly
fig = px.bar(top_10_stressful_professions, x='Occupation', y='Stress Level',
            title='Top 10 Professions with the Highest Stress Level',
            labels={'Occupation': 'Profession', 'Stress Level': 'Mean Stress Level'})

fig.show()
...

This code will generate a bar chart showing the top 10 professions with the highest mean stress level. The x-axis will display the profession, and the y-axis will display the mean stress level.
```

Fig. 7. Experiment № 1 – Generated script

5.2.2 Step 2 “Customization of a Previously Created Chart”

Task: “Change the color of bars to yellow, set the title to “Professions with the highest stress level”, and display only the top 5 professions”.

Response and commentary: the model successfully addressed all requirements, dynamically updating the visualization (Fig. 8). The chat history is updated to document the changes and interactions (Fig 9).

5.2.3 Step 3 “Complex Task with No Specified Chart Type”

Task: “Show me a visualization from which I can draw many conclusions”.

Response and commentary: given the open-ended nature of the task, the model produced a scatter plot (Fig. 10). This visualization illustrates the relationship between Sleep Duration, Physical Activity Level, and Stress Level for each individual. The system further provides interpretative questions (e.g., “Do individuals with higher Physical Activity Level tend to have lower Stress Levels or higher Daily Steps?”, “Are there any clusters or patterns in the data that could indicate underlying relationships between variables?”), thereby facilitating comprehensive analytical inquiry.

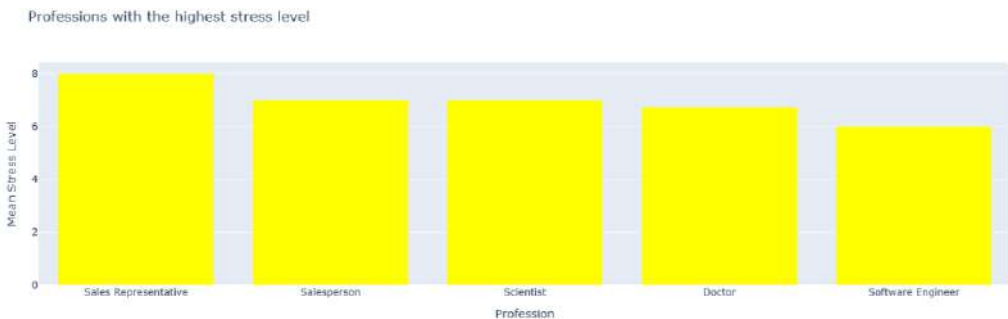


Fig. 8. Experiment № 2 – Generated visualization

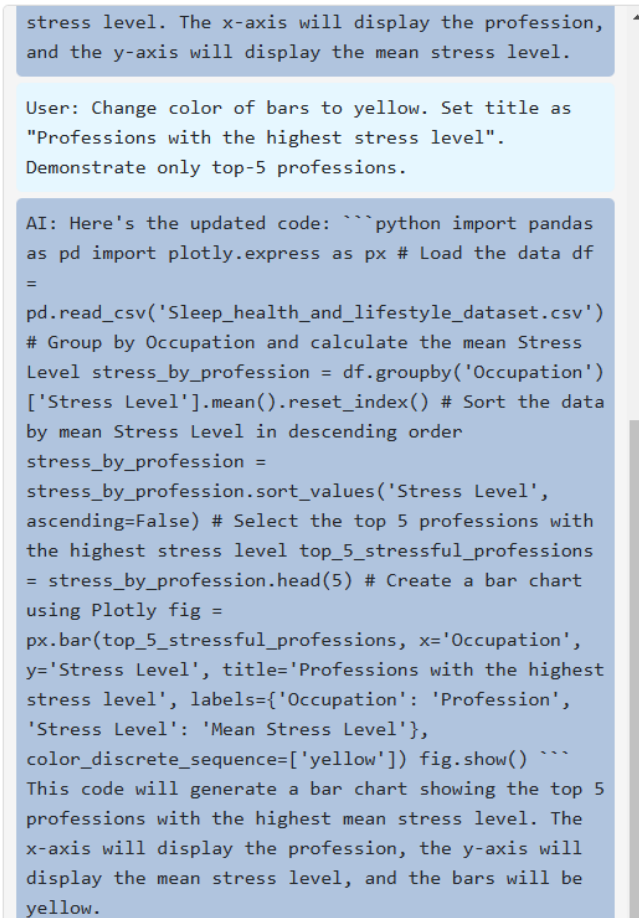


Fig. 9. Experiment № 2 – Chat history



Fig. 10. Experiment № 3 – Generated visualization

5.2.4 Step 4 “Predefined Chart Type, No Specific Concept”

Task: “Construct a pie chart”.

Response and commentary: in response, the model generated a pie chart to display the distribution of BMI categories within the dataset (Fig. 11).

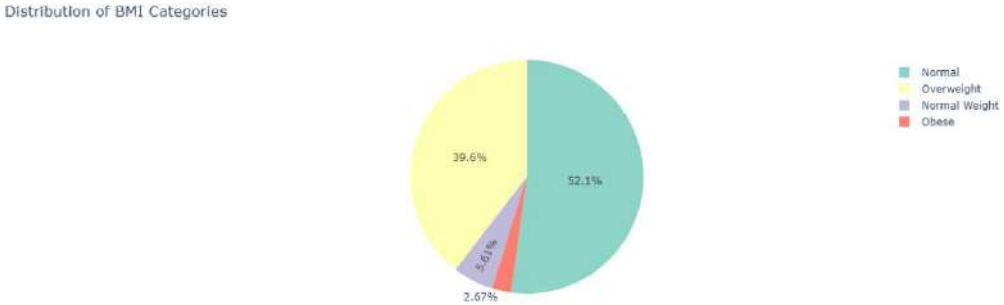


Fig. 11. Experiment № 4 – Generated visualization

5.3 Results of the Experiments: Summary

Collectively, these examples demonstrate a fraction of the visualization capabilities achievable with the AI-assistant. Moreover, an analysis of the ontology reveals that it has been dynamically updated throughout these interactions. The sequence of relationships “prompt – generated script – metadata” is preserved within the ontology individuals, confirming the robust integration of the knowledge-driven framework (Fig. 12).



Fig. 12. An example of Prompt class individual

6. Evaluation of the AI-assistant

The AI-assistant automates the creation and execution of Plotly scripts by transforming user’s natural-language requests into ready-to-run code. During testing, it demonstrated strong performance across a variety of tasks: from drawing basic bar charts to interpreting more abstract specifications. It also simplifies choosing the most appropriate visualization for a given dataset or analysis goal. By displaying the generated code, the assistant serves as a teaching aid for both Python and Plotly.

The prototype’s architectural novelty lies in integration with two core components:

1. A *multifaceted ontology* that logs every user request and generated script, enabling scalable knowledge accumulation, reuse of code templates across similar scenarios, and enhanced collaborative analytics.
2. A *library extension subsystem* that develops custom plugins atop existing libraries, trains the model to invoke these domain-driven functions natively, and in future iterations will leverage an *AIModelManager* to automatically select or update to the latest fine-tuned LLM version.

However, generative models remain prone to “hallucinations”, inventing nonexistent methods, producing syntactically incorrect code, or misinterpreting parameters, which can undermine script’s reliability. To mitigate these risks, the interface displays a persistent footer reminder to verify the assistant’s outputs and shows a clear “*Error when generating the graph*” notification if execution fails.

Currently, users are responsible for validating the generated code via these error notifications, because the system does not automatically check the accuracy of the model output against the user’s intent. To address this issue, future work includes training the model on real data and implementing performance criteria for effective visualizations. Derived from cognitive-visualization research [17, 18], these metrics will be formalized and embedded in the toolkit to enable automated compliance checks of visualizations against best practices.

7. Conclusion

In conclusion, this study has designed and developed a data visualization assistant as a part of a prototype of knowledge-driven data visualization tools. The innovative aspect of this approach lies in the integration of an AI-assistant with an ontology-based framework, which facilitates scalable knowledge accumulation. This integration significantly enhances collaborative analytics by enabling the sharing of standardized scripts and domain specific insights among user communities, thereby reducing redundancy and fostering robust reproducibility in data-driven workflows.

Future work will first expand the Plotly ecosystem with new custom packages and then train the model on user-defined modules to better support atypical visualizations. Criteria for effective visualizations [17, 18] will also be formalized and embedded into the platform to automatically analyze the compliance of constructed diagrams with best practices and help users avoid common mistakes.

References

- [1]. Sawicki J., Burdukiewicz M. VisQualdex: a Comprehensive Guide to Good Data Visualization. Scientific Visualization, 2023, vol. 15, no. 1, pp. 127–149. DOI: 10.26583/sv.15.1.11.
- [2]. Qin X., Luo Y., Tang N., Li G. Making Data Visualization More Efficient and Effective: A Survey. The VLDB Journal, 2020, vol. 29, no. 1, pp. 93–117. DOI: 10.1007/s00778-019 00588-3.
- [3]. Goldsberry K. Courtvision: New Visual and Spatial Analytics for the NBA. In Proc. of the 2012 MIT Sloan Sports Analytics Conference, 2012, pp. 12–15.
- [4]. Singh A. Democratizing Data Visualization and Insights Extraction with Pandas, Generative AI, and CSV Data. International Journal of Scientific Research in Engineering and Management, 2024, vol. 8, no. 5, pp. 1–5. DOI: 10.55041/ijrem33437.
- [5]. Dzheiranian A. D., Ermakov I. D., Proskuryakov K. A., Lyadova L. N. Designing Data Visualization System Based on Language-Oriented Approach. Proceedings of the Institute for System Programming of the RAS, 2024, vol. 36, no. 2, pp. 127–140. DOI: 10.15514/ISPRAS-2024-36(2)-10.
- [6]. Dzheiranian A. D., Ermakov I. D., Proskuryakov K. A., Lyadova L. N. Development of Data Visualization Tools Based on Domain Specific Modeling. In Proc. of the 34th International Conference on Computer Graphics and Machine Vision (GraphiCon 2024), 2024, pp. 300–314. DOI: 10.25206/978-5-8149-3873-2-2024 300-314.
- [7]. Dzheiranian A. D., Ermakov I. D., Proskuryakov K. A., Lyadova L. N. An Approach to Developing Data Visualization Tools Based on Domain Specific Modeling. Scientific Visualization, 2024, vol. 16, no. 4, pp. 82–101. DOI: 10.26583/sv.16.4.08.
- [8]. Shakeel H. M., Iram S., Al-Aqrabi H., Alsboui T., Hill R. A Comprehensive State-of-the-Art Survey on Data Visualization Tools: Research Developments, Challenges and Future Domain Specific Visualization Framework. IEEE Access, 2022, vol. 10, pp. 96581–96601. DOI: 10.1109/ACCESS.2022.3205115.
- [9]. Shi D., Xu X., Sun F., Shi Y., Cao N. Calliope: Automatic Visual Data Story Generation from a Spreadsheet. IEEE Transactions on Visualization and Computer Graphics, 2021, vol. 27, no. 2, pp. 453–463. DOI: 10.1109/TVCG.2020.3030403.

- [10]. Wang X., Wu Z., Huang W., Wei Y., Huang Z., Xu M., Chen W. VIS+AI: Integrating Visualization with Artificial Intelligence for Efficient Data Analysis. *Frontiers of Computer Science*, 2023, vol. 17, no. 6. DOI: 10.1007/s11704-023-2691-y.
- [11]. Wu A., Wang Y., Shu X., Moritz D., Cui W., Zhang H., Zhang D., Qu H. AI4VIS: Survey on Artificial Intelligence Approaches for Data Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 2021, vol. 28, no. 12, pp. 5049–5070. DOI: 10.1109/tvcg.2021.3099002.
- [12]. Maddigan P., Susnjak T. Chat2VIS: Generating Data Visualisations via Natural Language using ChatGPT, Codex and GPT-3 Large Language Models. *IEEE Access*, 2023, vol. 11, pp. 45181–45193. DOI: 10.1109/access.2023.3274199.
- [13]. Mallick T., Yildiz O., Lenz D., Peterka T. ChatVis: Automating Scientific Visualization with a Large Language Model. In *Proc. of the SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2024, pp. 49–55. DOI: 10.1109/SCW63240.2024.00014.
- [14]. Wang C., Lee B., Drucker S., Marshall D., Gao J. Data Formulator 2: Iteratively Creating Rich Visualizations with AI. *arXiv preprint*, 2024. arXiv: 2408.16119.
- [15]. Hong J., Seto C., Fan A., Maciejewski R. Do LLMs Have Visualization Literacy? An Evaluation on Modified Visualizations to Test Generalization in Data Interpretation. *IEEE Transactions on Visualization and Computer Graphics*, 2025. DOI: 10.1109/TVCG.2025.3536358.
- [16]. Alves T. FAIR Data: What It Is and How We Can Support Its Principles. *Science Editor*, 2024. DOI: 10.36591/se-4703-04.
- [17]. Midway S. R. Principles of Effective Data Visualization. *Patterns*, 2020, vol. 1, no. 9. DOI: 10.1016/j.patter.2020.100141.
- [18]. Midway S. R., Brum J. R., Robertson M. Show and Tell: Approaches for Effective Figures. *Limnology and Oceanography Letters*, 2023, vol. 8, no. 2, pp. 213–219. DOI: 10.1002/lol2.10288.

Информация об авторах / Information about authors

Анна Даниеловна ДЖЕЙРАНЯН – студент бакалавриата Национального исследовательского университета «Высшая школа экономики» (НИУ ВШЭ – Пермь), образовательная программа «Программная инженерия». Сфера научных интересов: анализ и визуализация данных, генеративные модели, предметно-ориентированное моделирование.

Anna Danielovna DZHEIRANIAN – undergraduate student at the National Research University – Higher School of Economics (HSE University, Perm Branch), educational program “Software Engineering”. Research interests: data analysis and visualization, generative models, domain specific modeling.

Людмила Николаевна ЛЯДОВА – кандидат физико-математических наук, доцент, доцент кафедры информационных технологий в бизнесе Национального исследовательского университета «Высшая школа экономики» (НИУ ВШЭ–Пермь). Сфера научных интересов: языки моделирования, предметно-ориентированное моделирование, языковые инструментари, CASE-средства, системы имитационного моделирования.

Lyudmila Nikolaevna LYADOVA – Cand. Sci. (Phys.-Math.) in Computer Science, Associate Professor of the Department of Information Technologies in Business of the National Research University – Higher School of Economics (HSE University, Perm Branch). Research interests: modeling languages, domain specific modeling, language toolkits, CASE tools, simulation systems.

DOI: 10.15514/ISPRAS-2025-37(4)-27



Development of Knowledge-based Intelligence for Sustainability Assessment of Russian Regions

¹ D.S. Fedoseev, ORCID: 0009-0003-0118-6425 <fs159@mail.ru>

² A.D. Neroslov, ORCID: 0000-0002-8870-1607 <9091069060@mail.ru>

¹ V.V. Lanin, ORCID: 0000-0002-0650-2314 <vlanin@hse.ru>

¹ HSE University,

38, Studencheskaia St., Perm, 614070, Russia.

² Perm National Research Polytechnic University,

29, Komsomolsky Prospekt, Perm, 614990, Russia.

Abstract. The paper presents the development of a Knowledge-based Intelligence for Sustainability Assessment (KISA) system for the comprehensive assessment of the sustainability of Russian regions, which uses a large language model (LLM) with retrieval-augmented generation (RAG) technology and Rosstat data. KISA automatically selects relevant indicators based on users' textual queries, determines their weights, and calculates regional ratings, overcoming the limitations of traditional methods associated with high resource costs, subjectivity, and low adaptability. The system reduces the time required for rating formation to 10 minutes – 140 times faster than existing approaches; financial costs are reduced by a factor of 16 due to the minimization of expert participation. The agreement with expert evaluations is 68%, confirming the validity of the method. KISA provides a web interface with map visualization, enhancing flexibility in analysis; the possibility of improvement through the addition of new sources ensures the continuous incorporating experts' experience. The results of the study contribute to the improvement of regional sustainability assessment and can be used in management decision-making.

Keywords: sustainability assessment; Russian regions; artificial intelligence; large language model LLM; retrieval-augmented generation RAG.

For citation: Fedoseev D.S., Neroslov A.D., Lanin V.V. Development of Knowledge-based Intelligence for Sustainability Assessment of Russian Regions. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025, pp. 207-218. DOI: 10.15514/ISPRAS-2025-37(4)-27.

Acknowledgements. This work was supported by the Ministry of Science and Higher Education of the Russian Federation (project No. FSNM-2024-0005).

Разработка интеллектуальной системы на основе знаний для оценки устойчивости российских регионов

¹ Д.С. Федосеев, ORCID: 0009-0003-0118-6425 <fs159@mail.ru>

² А.Д. Нерослов, ORCID: 0000-0002-8870-1607 <9091069060@mail.ru>

¹ В.В. Ланин, ORCID: 0000-0002-0650-2314 <vlanin@hse.ru>

¹ Национальный исследовательский университет «Высшая школа экономики»,
Россия, 614070, г. Пермь, ул. Студенческая, д. 38.

² Пермский национальный исследовательский политехнический университет,
Россия, 614990, г. Пермь, Комсомольский проспект, д. 29

Аннотация. В статье представлено описание процесса разработки системы интеллектуальной оценки устойчивости на основе знаний (KISA) для комплексной оценки устойчивости российских регионов, которая использует большую языковую модель (LLM) с технологией дополненной генерации поиска (RAG) и данные Росстата. KISA автоматически отбирает релевантные показатели на основе текстовых запросов пользователей, определяет их веса и рассчитывает региональные рейтинги, преодолевая ограничения традиционных методов, связанные с высокими ресурсными затратами, субъективностью и низкой адаптивностью. Система сокращает время, необходимое для формирования рейтинга, до 10 минут – в 140 раз быстрее существующих подходов; финансовые затраты снижаются в 16 раз за счет минимизации участия экспертов. Согласованность с экспертными оценками составляет 68%, что подтверждает валидность метода. KISA предоставляет веб-интерфейс с картографической визуализацией, повышая гибкость анализа; возможность совершенствования посредством добавления новых источников обеспечивает непрерывное включение экспертного опыта. Результаты исследования способствуют совершенствованию оценки региональной устойчивости и могут использоваться при принятии управленческих решений.

Ключевые слова: оценка устойчивости; российские регионы; искусственный интеллект; большие языковые модели LLM; технология RAG.

Для цитирования: Федосеев Д.С., Нерослов А.Д., Ланин В.В. Разработка интеллектуальной системы на основе знаний для оценки устойчивости российских регионов. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 207–218 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(4)–27.

Благодарности. Работа выполнена при поддержке Министерства науки и высшего образования Российской Федерации (проект № FSNM-2024-0005).

1. Introduction

The problem of sustainable development of the regions of the Russian Federation has been an urgent interdisciplinary task that requires a comprehensive approach. Regional sustainability is defined as the ability of a region to maintain the quality of life of the population, resist negative impacts, adapt to changes and utilize opportunities for long-term development [1-2].

According to the World Bank [3], Russia ranks third in terms of interregional inequality in Europe and Central Asia, which, in combination with Presidential Decree No. 309 “On the National Development Goals of the Russian Federation for the Period up to 2030 and in Perspective up to 2036” [4], emphasizes the need for a systematic regional sustainability assessment (SA) to reduce the socio-economic gap between regions. This conclusion is confirmed by the studies of Russian scientists [5-6].

At present, assessments are predominantly based on authors’ methods and expert approaches implemented through information systems (IS) for calculating indexes and compiling ratings of regions. Nevertheless, as Ramos notes [7], existing solutions face serious limitations.

The key problems include significant resource costs associated with attracting experts, as well as, the subjectivity and narrow focus of methods that often address a single area, ignoring a

comprehensive approach. Moreover, these systems show low adaptability to modern challenges such as climate change and geopolitical crises due to the rigid structure of the indicators used.

The solution to these problems is the development of the Knowledge-based Intelligence for Sustainability Assessment (KISA) system, which utilizes large language model (LLM) with retrieval-augmented generation (RAG) technology and Rosstat data for regional SA. Based on users' text queries, the system automatically selects relevant indicators, determines their weighting coefficients and calculates the integral rating of regions, overcoming the limitations of traditional methods.

2. Problem Statement

The aim of this research is to develop the KISA system capable of generating ratings of Russian regions based on individual user requests, thereby enhancing the objectivity of assessment while reducing the resource intensity of the process.

The project considers only the constitutionally enshrined constituent entities of the Russian Federation (as of 2020), with the analysis based on Rosstat's open data on environmental, economic, and social indicators for the period 2000-2024. While Rosstat provides the most comprehensive official statistics available, it is important to acknowledge potential limitations such as reporting delays and methodological changes, though these constraints remain consistent across all analytical approaches to regional assessment.

Project success criteria:

- Achieving at least an 80% similarity with expert assessments using quality metrics.
- Reducing resource costs for assessment: time by a factor of 7, finances by a factor of 5 compared to traditional expert methods.

The KISA system will not only ensure increased efficiency, adaptability and objectivity of regional SA, but will also simplify the decision-making process for stakeholders, such as government agencies, think tanks and business structures.

3. Related Works

This section focuses on ways to overcome the identified limitations: enhancing the adaptability of evaluation criteria, ensuring objectivity, and optimizing costs compared to traditional approaches. Special attention is paid to the prospects of using artificial intelligence (AI) technologies to address these problems.

3.1 Expert Methods for Sustainability Assessment

Methods for SA usually include sets of statistical indicators and algorithms for calculating an integral index reflecting the level of sustainability of the territory. Numerous such methods have been proposed by both foreign researchers [8-10] and Russian researchers [11-13]. However, many of them have disadvantages: low adaptability due to the use of static indicators and high resource consumption, requiring large research teams.

A methodological review by Lindfors [14] examined the biases that arise when applying such methods, where an excess or deficiency of criteria can distort the evaluation results. This emphasizes the need to develop more efficient, objective and adaptive approaches.

The next subsection examines existing IS for SA, most of which are based on the methods described previously, thus inheriting their limitations.

3.2 Information Systems for Sustainability Assessment

IS for SA were analyzed to identify their strengths and weaknesses for project development.

Three systems were considered: SberAnalytics "Monitoring of regional economies" [15], which provides economic indicators for analyzing regions; Foresight Analytics Platform [16], which has a

modular architecture and analytical models; CSA-system [17], which focuses on environmental sustainability of buildings.

These IS applied expert methods for SA, which resulted in the problems of resource intensity, subjectivity and low flexibility. For example, SberAnalytics requires regular participation from experts to customize indicators, and the rigid set of these indicators makes it difficult to adapt to new conditions. Although Foresight allows customization of indicators, the final decisions still depend on expert opinion.

Previously, the current authors (Fedoseev et al.) developed an IS for regional SA [18], which also inherited the limitations of expert methods: the user must independently select indicators and set weighting coefficients, thus preserving the outlined problems.

Despite the advantages of IS in data processing [19], existing solutions only partially solve the problems. The development of an intelligent system is needed to overcome these limitations. The use of AI techniques is promising for creating an effective SA tool [20].

3.3 Application of Artificial Intelligence for Sustainability Assessment

The application of AI in the SA area significantly improves the efficiency of solutions [21]. In particular, the use of Russian-adapted LLMs such as RuAdaptQwen-2.5 [22] with the Chain of Thought approach [23] will allow the KISA system to flexibly customize criteria based on users' textual queries, thereby increasing the adaptability of the system.

The addition of RAG to the LLM will provide the KISA system with the ability to utilize validated data when selecting indicators and setting weights, ensuring a high degree of objectivity in SA. Studies in related fields confirm the effectiveness of this approach: Bronzini et al. [24] applied it to extract information from sustainability reports, and Arslan et al. [25] developed a chatbot for the sustainable energy transition. However, an integrated AI-based regional SA system has not yet been developed.

Systems combining LLM and RAG show significant superiority over traditional methods. According to a study by Ren et al. [26], LLMs outperform humans in all environmental and economic metrics, reducing costs by more than 150 times (from \$12.1 to \$0.08), which solves the problem of high resource costs associated with expert involvement.

Thus, integrating LLM and RAG into the KISA system will create the first comprehensive system for regional SA, overcoming the key limitations of traditional methods: low adaptability, subjectivity, and high resource costs.

4. System Design

For the systematized storage and processing of statistical data obtained from Rosstat, a database in the form of an ER-diagram was designed (Fig. 1). The central entity "indicator" links numerical indicators with a specific region and time period. The "indicator_type" entity contains the names of specific indicators (e.g., "Unemployment rate"), which are categorized through the "indicator_type_group" entity (e.g., "Economic sphere"). The entity "external_service_data" allows for the integration of information from external sources.

The KISA system is designed using a modular multi-layered architecture with clear separation of responsibilities between components: web interface, controllers, services, and repositories. This approach ensures independent modules and facilitates maintenance and scalability of the application. The C4 model, which divides the system into Context, Container, Component and Code levels, was used as a design methodology [27].

The component-level architecture is represented in the diagram (Fig. 2), while the detailed description of the main modules and their functions is provided in Table 1.

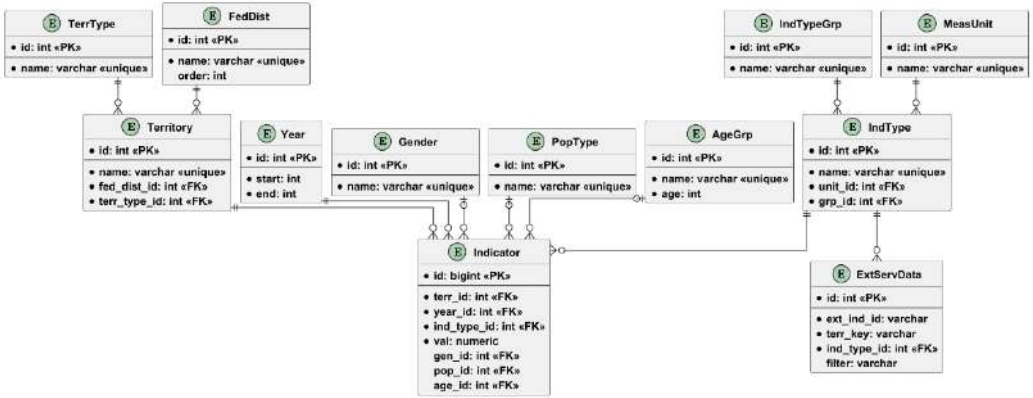


Fig. 1. Entity-Relationship Model of the Database.

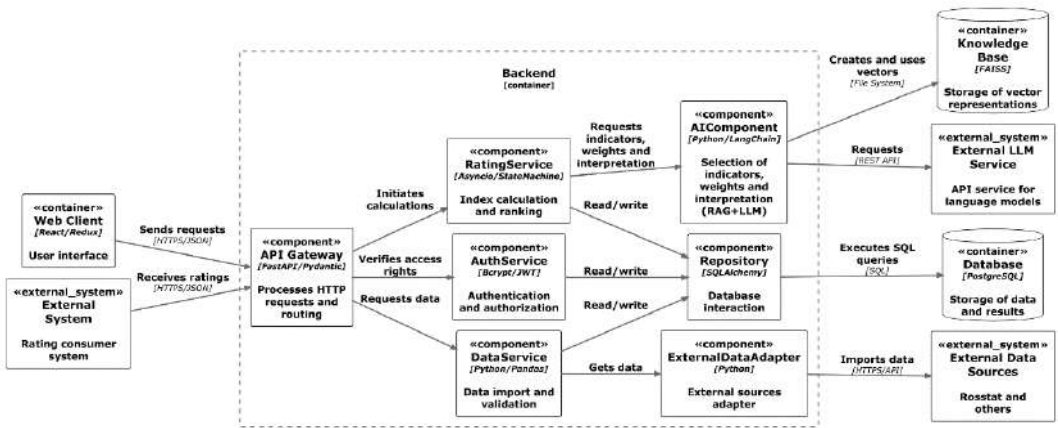


Fig. 2. Backend Components Architecture in C4 Model.

Table 1. Main Modules and Functions.

Module	Functionality
Web Client	User Interface in React
API Gateway	HTTP request processing and routing
AuthService	User authentication and authorization
DataService	Import, validation and structured storage of statistical data using ETL approach
RatingService	Data normalization, removal of outliers, calculation of indexes by weighted sum of indicators and compilation of region rankings
AIComponent	Dynamic selection of relevant indicators and their weights based on text queries using integrated LLM and RAG technologies
Repository	Data access abstraction and database interaction

DataService uses the Extract, Transform and Load (ETL) approach to integrate open statistical data of Rosstat, which ensures a uniform format and correctness of input data [28]. This is important for data unification and subsequent analysis, although possible incompleteness or inaccuracy of the original information should be taken into account.

The algorithm for generating the ratings includes (Fig. 3):

1. User input of a text query defining the SA parameters.
2. Application of RAG technology to extract relevant scientific data from the knowledge base.
3. Transmission of the context, the user's query, and the full list of indicators to the LLM.
4. Automatic selection of relevant indicators by the LLM and determination of their weighting coefficients.
5. Extraction of selected statistical data from PostgreSQL and calculating the integral rating of regions.
6. Visualization of the results with cartographic representation.

The result is a rating reflecting the relative position of each region in terms of regional sustainability.

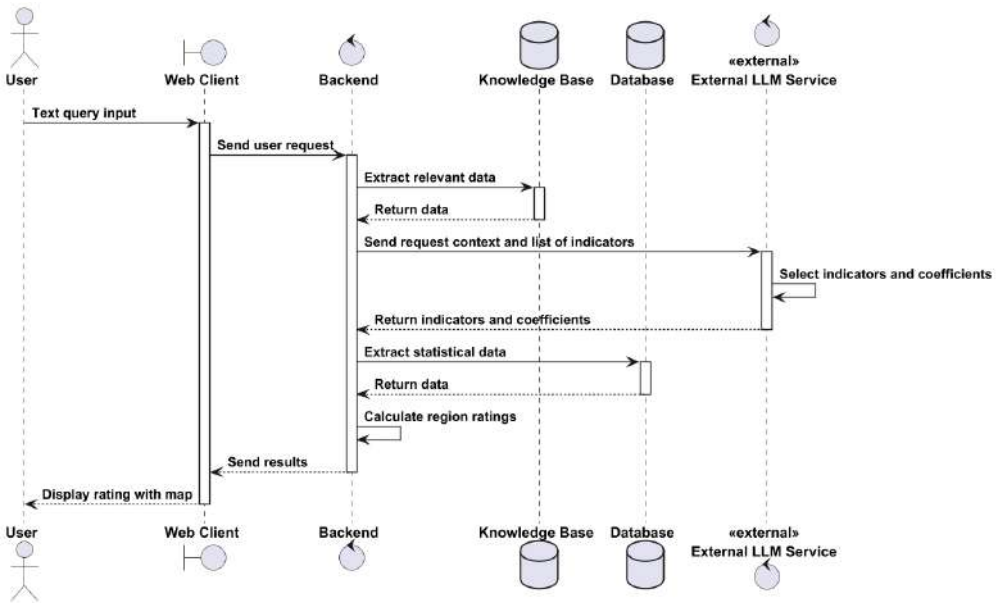


Fig. 3. Rating Formation Process.

5. Implementation

For the development of the KISA system prototype, a technology stack was chosen, including Python, FastAPI for creating a RESTful API [29], and SQLAlchemy for working with a PostgreSQL database. The frontend is developed using React to create an interactive user interface with dynamic graphs and map-based visualization.

LangChain and OpenAI libraries are used to implement RAG and LLM methods. The open-source RuAdaptQwen-2.5 (32 billion parameters), which uses the Chain-of-Thought approach to improve query efficiency, was chosen as the LLM. However, the use of such models requires significant computational power, which may limit query-processing speed. This limitation can be overcome by using powerful servers.

The developed prompts follow effective engineering principles: clear task definition, structured input data presentation, explicit step-by-step reasoning requirements, and formalized output format

(Fig. 4). The system prompt defines the model's role as a data analysis expert and establishes basic operational principles. The user prompt includes a specific task (selecting up to a given number of indicators), user query, knowledge base context, available indicators list, and required response format with examples.

```
## Task:
1. Select up **to {max_inds}** indicators that correspond
to the query.
2. Specify the influence: **1** (positive) or **-1**
(negative).
3. Distribute weights so that the most significant
indicators receive the highest weight, and the least
significant ones receive the lowest.

## User query:
{query}
{context_block}
## List of indicators with ID:
{indicators_text}

## Response format:
1. Explain your choice and distribution of influence and
weights step by step.
2. Provide a JSON response. Example:
```json
{example}
```
```

Fig. 4. Prompt Template Structure for Indicator Selection

For the RAG knowledge base, publications from leading socio-economic journals indexed in Scopus and Web of Science were selected. A total of 243 scientific journals were loaded, covering a wide spectrum of research in regional economics, sustainable development, and methods for assessing territorial entities. The publications were cleaned using regular expressions and segmented into chunks, resulting in 398,290 text fragments. These fragments were then stored in the FAISS vector database using embeddings [30].

KISA integrates RAG technology to access scientific literature for regional sustainability assessments, expanding domain knowledge without costly retraining. By retrieving information from published research, the system bases indicator selection on established methodologies rather than pre-trained knowledge. This integration addresses three critical challenges: resource intensity is reduced through automated analysis, adaptability improves as the LLM flexibly selects indicators based on user text queries, and subjectivity decreases as evaluations rely on published research rather than individual judgment.

6. Evaluation

For the evaluation of the KISA system prototype, an expert commission was formed. The commission consisted of 11 members, including 7 Doctors of Science and 4 Candidates of Science in economics, engineering, history, and social sciences. The commission was tasked with creating three text-query assessment methods based on a set of indicators and the weighted-sum formula used in the KISA system.

The results of the expert assessment and the KISA system were compared. The Recall metric (the ratio of the number of matching indicators to the number of indicators selected by the expert) was used to measure the similarity of the selected indicators, and the MAPE (Mean Absolute Percentage Error) metric, which evaluates the average percentage error between the weights, was used to compare the indicator weights.

The average similarity of the results was 68% (Fig. 5). It is worth noting that a 100% match was not expected, as it is impossible to claim that one method is unequivocally better than the other. The indicators chosen by KISA and experts were similar in meaning but differed in naming. For example, the expert commission chose “Level of education”, while KISA chose “Growth rate of education level”. Although the scope of assessment of the indicators is the same, the approaches to assessment differ, so the results are not completely the same.

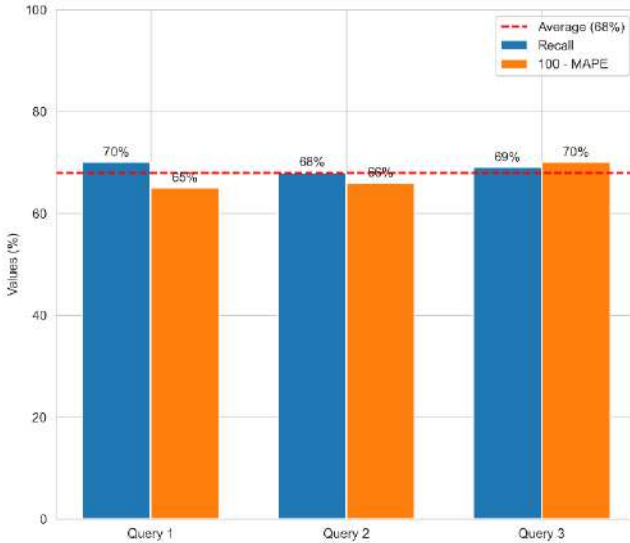


Fig. 5. Comparison of Recall and MAPE across Queries

A straightforward comparison of the system and the experts is complicated, as their assessments may be subjective and reflect different points of view. The similarity of the results also depends on the composition of the expert commission, the specific query, and the content of the knowledge base.

Now, consider comparing the resource costs of the KISA system and the expert commission. According to HeadHunter, an hour of an expert’s work costs 1,000 rubles. An expert commission of 11 people spent three days (24 hours) to create one method, which amounted to 264,000 rubles and required the participation of the entire group. The development of the KISA system took four months with a workload of 160 hours per month. The average salary of a software engineer is 100,000 rubles per month, so the costs totaled about 400,000 rubles.

KISA generates regional ratings on request in 10 minutes, including data input, which saves both financial and time resources. Time costs are reduced by a factor of 140: experts spent 1,440 minutes, and the system completes the same work in 10 minutes. This allows the system to pay for itself after the first request.

Financial costs are reduced 16-fold when using KISA. An expert commission can create a maximum of six methods per month, which costs 1.58 million rubles. At the same time, KISA’s maintenance and infrastructure costs are about 100,000 rubles per month. Thus, the use of the KISA system significantly reduces both time and financial costs, ensuring the efficiency and accessibility of the tool for regional SA.

7. Conclusion

As a result of this study, a working prototype of the KISA system has been developed that addresses three key problems of existing approaches:

1. Resource intensity. Automation of data collection and processing through ETL pipelines and the use of the RuAdaptQwen-2.5 LLM has reduced the time required for rating generation to 10 minutes, which is 140 times faster. Financial costs are reduced 16-fold due to the minimization of expert involvement.
2. Low adaptability. The system interprets text queries in natural language and adapts criteria for specific analysis purposes, providing flexibility and aligning with user needs.
3. Subjectivity. Automated selection of indicators and their weights based on RAG technology eliminates subjective expert assessments. A transparent algorithm based on scientific data increases the reproducibility and validity of the results. The average similarity between the results and expert assessments is 68%.

KISA facilitates the formation of regional ratings by calculating the final regional sustainability index and ranking, as well as visualizing the results through an interactive web interface with a map display.

The key feature of KISA is the possibility of continuous improvement by adding new scientific sources to the knowledge base, allowing the system to be updated and incorporating experts' experience.

The KISA system's database is registered in Rospatent [31]. The computer program has also been registered and received the corresponding certificate [32].

In the future, plans include introducing the function of forecasting sustainability indicators to provide a complete picture of regional development for the next 10 years. Moreover, additional formulas for calculating the sustainability index are expected to be implemented.

The obtained results are important for improving the process of regional SA and can be used to support managerial decisions. Further research will be aimed at expanding the functionality of the system and updating it with new data.

References

- [1]. Перфилов В. А. Сущность и типы устойчивости развития региональных социально-экономических систем // ПСЭ. №2., 2012 г., стр 264-266. / Perfilov V. The essence and types of sustainability of regional socio-economic systems. *Problems of the modern economy*, 2012, vol. 2, pp. 264-266 (In Russian).
- [2]. Jovovic R., Draskovic M., Delibasic M., Jovovic M. The concept of sustainable regional development—institutional aspects, policies and prospects. *Journal of International Studies*, 2017, vol. 10, No. 1, pp. 255-266.
- [3]. Inside the World Bank's new inequality indicator: The number of countries with high inequality. [Online]. Available at: <https://blogs.worldbank.org/en/opendata/inside-the-world-bank-s-new-inequality-indicator-the-number-of->, accessed: 22 May 2025.
- [4]. О национальных целях развития Российской Федерации на период до 2030 года и на перспективу до 2036 года: Указ Президента РФ от 7 мая 2024 г. № 309 / Decree of the President of the Russian Federation. No. 309. "On the national development goals of the Russian Federation for the period up to 2030 and for the future up to 2036" Available at: <https://mvd.consultant.ru/documents/1058493>, accessed: 22 May 2025 (In Russian).
- [5]. Mareeva S. Socio-economic inequalities in modern Russia and their perception by the population. *The Journal of Chinese Sociology*, 2020, vol. 7, No.1, 10 p.
- [6]. Shatalova O., Kasatkina, E. Socio-economic inequality of regions in the Russian Federation: Measurement issues and long-term evaluation. *Economic and Social Changes: Facts, Trends, Forecast*, 2022, vol. 15, No. 4, pp. 74-87.
- [7]. Ramos T. Sustainability Assessment: Exploring the Frontiers and Paradigms of Indicator Approaches. *Sustainability*, 2019, vol. 11, No. 3, 824 p.
- [8]. Zhong R., Pei F., Yang K., Xia Y., Wang H., & Yan G. Coordinating socio-economic and environmental dimensions to evaluate regional sustainability—towards an integrative framework. *Ecological Indicators*, 2021, vol. 130, 108085 p.
- [9]. D'Adamo I., Falcone P., Imbert E., Morone P. Exploring regional transitions to the bioeconomy using a socio-economic indicator: The case of Italy. *Economia Politica*, 2022, vol. 39, No. 3, pp. 989-1021.

- [10]. Ziegler D., Wolff S., Agu A.-B., Cortiana G., Umair M., Durfort F., Neumann E., Walther G., Kristiansen J., Lienkamp M. How to Measure Sustainability? An Open-Data Approach. *Sustainability*, 2023, vol. 15, No. 4, 3203 p.
- [11]. Швец И. Ю., Швец Ю. Ю., Чиж-Гвезда Э. Оценка устойчивого инновационного развития региона // Вестник Ассоциации вузов туризма и сервиса. том 9. № 1. 2015 г., стр. 14-21. / Shvec I., Shvec Ju., Chizh-Gvyezda Je. Assessment of the sustainable innovative development of the region. *Bulletin of the Association of Universities of Tourism and Service*, 2015, vol. 9, pp. 14-21 (In Russian).
- [12]. Яшина Н. И., Яшин С. Н., Вилейшикова А. А. Методический инструментарий оценки социально-экономической безопасности регионов на основе формирования системы целевых показателей производственной и непроизводственной сфер развития регионов. Вестник Нижегородского университета им. Н.И. Лобачевского. Серия: Социальные науки. Вып. 2, № 62, 2021 г., стр. 45-54. DOI: 10.52452/18115942_2021_2_45. / Jashina N. I., Jashin S. N., Vilejshikova A. A. Methodological tools for assessing the socio-economic security of regions based on the formation of a system of target indicators for industrial and non-industrial areas of regional development. *Bulletin of the Nizhny Novgorod Lobachevsky University*, 2021, vol. 2, No. 62, pp. 45-54 (In Russian). DOI: 10.52452/18115942_2021_2_45.
- [13]. Бородин С. Н. Модель оценки устойчивого развития региона на основе индексного метода // Экономика региона. том 19, № 1, 2023 г., стр. 45-59. DOI 10.17059/ekon.reg.2023-1-4. / Borodin S. A model for assessing the sustainable development of a region based on the index method. *Economy of the region*. 2023, vol. 19, No. 1, pp. 45-59 (In Russian). DOI 10.17059/ekon.reg.2023-1-4.
- [14]. Lindfors A. Assessing sustainability with multi-criteria methods: A methodologically focused literature review. *Environmental and Sustainability Indicators*, 2021, vol. 12, 100149 p.
- [15]. Сбер Аналитика: Мониторинг экономики региона / SberAnalytics “Monitoring of regional economies”, Available at: <https://sberanalytics.ru/products/gossector/mr>, accessed: 22 May 2025 (In Russian).
- [16]. Форсайт. Аналитическая платформа / Foresight Analytics Platform, Available at: <https://www.fsight.ru/platform/>, accessed: 22 May 2025 (In Russian).
- [17]. Olawumi T., Chan D. Cloud-based sustainability assessment (CSA) system for automating the sustainability decision-making process of built assets. *Expert Systems with Applications*, 2022, vol. 188, 116020 p.
- [18]. Федосеев Д. С., Нерослов А. Д., Ланин В. В. Разработка информационной системы для оценки устойчивости регионов России. Сборник материалов III студенческой научно-практической конференции им. Л.Л. Любимова. 2025 г., стр. 170-175. / Fedoseev D., Neroslov A., Lanin V. Development of an Information System for Assessing the Sustainability of Russian Regions. Collection of materials of the III student scientific and practical conference named after L.L. Lyubimov at HSE, 2025, pp. 170-175, Available at: https://perm.hse.ru/editorial_publishing/Lyubimov_Conference3, accessed: 22 May 2025 (In Russian).
- [19]. Oliveira V., Teixeira D., Rocchi L., Boggia, A. Geographic Information System Applied to Sustainability Assessments: Conceptual Structure and Research Trends. *ISPRS International Journal of Geo-Information*, 2022, vol. 11, No. 11, 569 p.
- [20]. Arfanuzzaman M. Harnessing artificial intelligence and big data for SDGs and prosperous urban future in South Asia. *Environmental and sustainability indicators*, 2021, vol. 11, 100127 p.
- [21]. Greif L., Röckel F., Kimmig A., Ovtcharova J. A systematic review of current AI techniques used in the context of the SDGs. *International Journal of Environmental Research*, vol. 19, No. 1, 2025.
- [22]. Tikhomirov M., Chernyshev, D. Improving Large Language Model Russian adaptation with preliminary vocabulary optimization. *Lobachevskii Journal of Mathematics*, 2024, vol. 45, No. 7, pp. 3211-3219.
- [23]. Kojima T., Gu S., Reid M., Matsuo Y., Iwasawa Y. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 2022, vol. 35, pp. 2219-22213.
- [24]. Bronzini M., Nicolini C., Lepri B., Passerini A., Staiano J. Glitter or gold? Deriving structured insights from sustainability reports via large language models. *EPJ Data Science*, 2024, vol. 13, No. 41.
- [25]. Arslan M., Mahdjoubi L., Munawar, S. Driving sustainable energy transitions with a multi-source RAG-LLM system. *Energy and Buildings*, 2024, vol. 324, 114827 p.
- [26]. Ren S., Tomlinson B., Black R., Torrance A. Reconciling the contrasting narratives on the environmental impact of large language models. *Scientific Reports*, 2024, vol. 14, No. 1, 26310 p.
- [27]. Vázquez-Ingelmo A., García-Holgado A., García-Peñalvo, F. C4 model in a software engineering subject to ease the comprehension of uml and the software. In 2020 IEEE Global Engineering Education Conference, 2020, pp. 919-924.

- [28]. Nwokeji J., Matovu R. A systematic literature review on big data extraction, transformation and loading (etl). In *Intelligent Computing: Proceedings of the 2021 Computing Conference*, 2021, vol. 2, pp. 308-324.
- [29]. Narayanan P. Engineering Machine Learning and Data REST APIs using FastAPI. In *Data Engineering for Machine Learning Pipelines: From Python Libraries to ML Pipelines and Cloud Platforms*, 2024, pp. 323-359.
- [30]. Douze M., Guzhva A., Deng C., Johnson J., Szilvasy G., Mazaré P.-E., Lomeli M., Hosseini L., & Jégou H. The faiss library. arXiv preprint. arXiv:2401.08281, unpublished.
- [31]. Паздникова Н. П., Нерослов А. Д., Федосеев Д. С. Комплексная оценка территориальной устойчивости и потенциалов (КОТУП): свидетельство о государственной регистрации базы данных № 2024621915 Российская Федерация; № 2024620239; 2024 г./ Pазdnikova N. P., Neroslov A. D., Fedoseev D. S. Certificate of State registration of the database No. 2024621915 Russian Federation. Comprehensive assessment of territorial stability and potentials (CATSP), 2024.
- [32]. Нерослов А. Д., Федосеев Д. С., Паздникова Н. П. Комплексная оценка территориальной устойчивости и потенциалов (КОТУП): свидетельство о государственной регистрации программы для ЭВМ № 2025617985 Российская Федерация. 2025 г. / Neroslov A. D., Fedoseev D. S., Pазdnikova N. P. Certificate of state registration of the computer program No. 2025617985 Russian Federation. Comprehensive Assessment of Territorial Sustainability and Potentials (CATSP), 2025.

Информация об авторах / Information about authors

Данил Сергеевич ФЕДОСЕЕВ – выпускник бакалавриата Национального исследовательского университета «Высшая школа экономики», образовательная программа «Программная инженерия». Сфера научных интересов: обработка естественного языка, большие языковые модели, retrieval-augmented generation.

Danil Sergeevich FEDOSEEV – graduate of the Bachelor’s program in Software Engineering at the National Research University – Higher School of Economics. Research interests: natural language processing, large language models, retrieval-augmented generation.

Алексей Дмитриевич НЕРОСЛОВ – студент первого курса магистратуры Пермского национального исследовательского политехнического университета, образовательная программа «Государственное и муниципальное управление». Сфера научных интересов: экономика, экономика регионов России, социально-экономические показатели деятельности субъектов Российской Федерации.

Alexey Dmitrievich NEROSLOV – first-year master’s student at Perm National Research Polytechnic University in the State and Municipal Administration program. Research interests: economics, regional economics of Russia, socio-economic indicators of the performance of the constituent entities of the Russian Federation.

Вячеслав Владимирович ЛАНИН – старший преподаватель кафедры информационных технологий в бизнесе Национального исследовательского университета «Высшая школа экономики». Сфера научных интересов: языки моделирования, предметно-ориентированное моделирование, языковые инструментарии, CASE-средства, системы имитационного моделирования.

Viacheslav Vladimirovich LANIN – Senior Lecturer of the Department of Information Technologies in Business of the National Research University – Higher School of Economics. Research interests: modeling languages, domain specific modeling, language toolkits, CASE tools, simulation systems.

DOI: 10.15514/ISPRAS-2025-37(4)-28



Combining Logical Reasoning and LLMs Toward Creating Multi-Agent Smart Home Systems

L. Rezunik, ORCID: 0009-0000-9428-4718 <lrezunik@hse.ru >

M.A. Prozorskiy, ORCID: 0009-0006-9250-7280 <mprozorskiy@hse.ru>

D.V. Alexandrov, ORCID: 0000-0002-9759-8787 <dvaalexandrov@hse.ru>

HSE University,

11, Pokrovsky boulevard, Moscow, 109028, Russia.

Abstract. The rapid advancement of AI technologies, particularly Large Language Models (LLMs), has sparked interest in their integration into Multi-Agent Systems (MAS). This holds substantial promise for applications such as smart homes, where it can significantly enhance user experience by optimizing comfort, energy efficiency, and security. Despite the potential benefits, the implementation of MAS based on LLMs faces several challenges, including the risks of hallucinations, scalability issues, and concerns about the reliability of these systems in real-world applications. This study explores the development of MAS incorporating LLMs, with a focus on mitigating hallucinations through the integration of formal logical models for knowledge representation and decision-making, along with other machine learning methods. To demonstrate the efficacy of this approach, we conducted experiments with a plant care module within a smart home system. The results show that our approach can significantly reduce hallucinations and enhance the overall reliability of the system. Further research will focus on refining these methods to enhance adaptability and scalability to ensure system's functionality in real-world environments.

Keywords: multi-agent system; LLM-MA; first order logic; smart home.

For citation: Rezunik L., Prozorskiy M.A., Alexandrov D. V. Combining Logical Reasoning and LLMs Toward Creating Multi-Agent Smart Home Systems. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 4, part 2, 2025, pp. 219-234. DOI: 10.15514/ISPRAS-2025-37(4)-28.

Acknowledgements. This work is an output of a research project implemented as part of the Basic Research Program at the National Research University Higher School of Economics (HSE University).

Комбинирование логических рассуждений и LLM на пути к созданию мультиагентных систем умного дома

Л. Резуник, ORCID: 0009-0000-9428-4718 <lrezunik@hse.ru >

М.А. Прозорский, ORCID: 0009-0006-9250-7280 <mprozorskiy@hse.ru>

Д.В. Александров, ORCID: 0000-0002-9759-8787 <dval Alexandrov@hse.ru>

*Национальный исследовательский университет «Высшая школа экономики»,
Россия, 109028, г. Москва, Покровский бул., 11, стр. 10.*

Аннотация. Стремительное развитие технологий искусственного интеллекта, в частности больших языковых моделей (LLM), вызвало интерес к их интеграции в мультиагентные системы (МАС). Это открывает широкие перспективы в том числе для приложений умного дома, где они могут значительно улучшить пользовательский опыт за счет комфорта, энергоэффективности и безопасности. Несмотря на потенциальные преимущества, реализация МАС на основе LLM сталкивается с рядом проблем, включая риск возникновения галлюцинаций, проблемы масштабируемости и опасения по поводу надежности этих систем в реальных приложениях. В данном исследовании рассматривается разработка МАС, включающих LLM, с акцентом на уменьшение галлюцинаций путем интеграции формальных логических моделей для представления знаний и принятия решений, а также методов машинного обучения. Чтобы продемонстрировать эффективность этого подхода, были проведены эксперименты с симуляцией системы ухода за растениями в контексте умного дома. Результаты показали, что наш подход позволяет значительно уменьшить количество галлюцинаций и повысить общую надежность системы. Дальнейшие исследования будут направлены на доработку этих методов с целью повышения адаптивности и масштабируемости для обеспечения функциональности системы в реальных условиях.

Ключевые слова: мультиагентные системы; большие мультиагентные языковые модели LLM-MA; логика первого порядка; умный дом.

Для цитирования: Резуник Л., Прозорский М.А., Александров Д.В. Комбинирование логических рассуждений и LLM на пути к созданию мультиагентных систем умного дома. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 219–234 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(4)–28.

Благодарности. Данная работа является результатом исследовательского проекта, реализованного в рамках Программы фундаментальных исследований НИУ ВШЭ.

1. Introduction

Multi-Agent Systems (MAS) are a well-established concept in the field of distributed artificial intelligence, developed to enable multiple autonomous agents to operate, interact, and make decisions within a shared environment. These systems are designed to tackle complex problems that are difficult or inefficient for a single agent to solve alone [1]. MAS have been applied across various domains, including smart homes, industrial automation, robotics, and environmental simulation.

With the emergence of large language models (LLMs) and ongoing advancements in artificial intelligence, a trend has emerged toward integrating AI with MAS to enhance decision-making. As a result, the concept of LLM-MA (LLM Multi-Agent) has appeared – MAS that are either entirely based on LLM agents or incorporate LLMs. The relevance of such systems is supported by various studies showcasing their application in problem solving, world simulation, knowledge acquisition, and more [2].

However, since LLM-MA systems are based on LLMs, they inherit many of the same challenges: scalability issues, limited collaboration capabilities, and difficulty in assessing accuracy and reliability. The most significant problem is hallucination, which in MAS can lead to unpredictable behavior. Therefore, although LLMs can have a transformative impact on MAS development, many problems remain unsolved or only partially addressed, offering substantial opportunities for research.

In this paper, we focus on the integration of LLMs into MAS and the development of a hardware-software system for automated plant care as part of a smart home MAS. The goals of this research are: (1) to describe a novel approach aimed at reducing the effect of LLM hallucinations in MAS, (2) to propose a MAS architecture designed for the same purpose, (3) to demonstrate how this approach can be applied in a real-world smart home scenario focused on plant care.

The remainder of the paper is organized as follows: Section 2 introduces the methods underlying our approach. Section 3 and 4 detail the developed algorithms and the approach itself, showcasing a smart home plant-watering agent, its implementation, and experimental results. The final sections discuss the related work, results, and directions for future research.

2. Key Concepts and Methods

As mentioned previously, Multi-Agent Systems (MAS) are systems composed of agents – computational units designed to operate independently and efficiently in dynamic environments [3]. We can formalize the concept of a MAS and define it as a tuple:

$$MAS = (A, E, I)$$

Here A represents a set of agents, E – the environment, in which the system operates and I – a set of relationships between agents of MAS.

An agent in a multi-agent system can be described in terms of its actions, perceived environmental changes, and its responses – i.e., actions performed in reaction to those changes:

$$A = (Env, Act, R)$$

- Env – the set of observations the agent can receive from the environment.
- Act – the set of possible actions.
- R – the reaction function $R: Env \rightarrow Act$, which determines the action the agent takes based on the environment.

Agents may also possess internal states, and their actions can depend on these states. However, agents are generally considered reactive.

When discussing LLM agents in MAS, the environment of such an agent is defined by its inputs: user prompt, system prompt (hidden prompt), embeddings, and configuration settings (e.g., temperature, max tokens). All these elements can be controlled. However, it is difficult to explicitly define the actions of an LLM agent in response to a specific environmental trigger. Therefore, to reduce the effect of hallucinations, we can only manipulate the agent’s environment.

There are various machine learning methods developed to achieve this, as well as alternative approaches outside the ML domain – such as logical reasoning, which we will discuss later in this section. We will consider all relevant methods in the following subsections.

2.1 Machine Learning Methods

Many different machine learning methods have been developed to reduce hallucinations in LLMs. In this section, we will consider some of the most used approaches.

- *In-context Learning.*

This method encompasses all prompting techniques that involve providing the LLM with examples of similar tasks directly within the prompt. It is widely used due to its cost-effectiveness and as an alternative to fine-tuning.

There are various in-context learning techniques, such as few-shot learning, where several example input-output pairs are included in the prompt, and Chain-of-Thought (CoT) prompting [4], which encourages the LLM to describe its reasoning process and break the task into smaller, manageable steps.

- *Multi-Agent Debate.*

This method is specifically designed to mitigate hallucinations or misconceptions through iterative discussions among multiple agents [5-6] (no fewer than two). The core idea is that the same question is posed across several iterations until the agents reach a consensus. Once agreement is achieved, the debate ends, and the final answer is accepted as the truth.

However, this method is not always reliable. Depending on the model, one agent may be easily influenced by another, leading to biased or incorrect answers. To address this, knowledge bases are sometimes introduced into the debate. These are used by agents to support their responses, forcing the LLM to consider factual information before answering. This introduces a form of learning into the process (see Fig. 1).

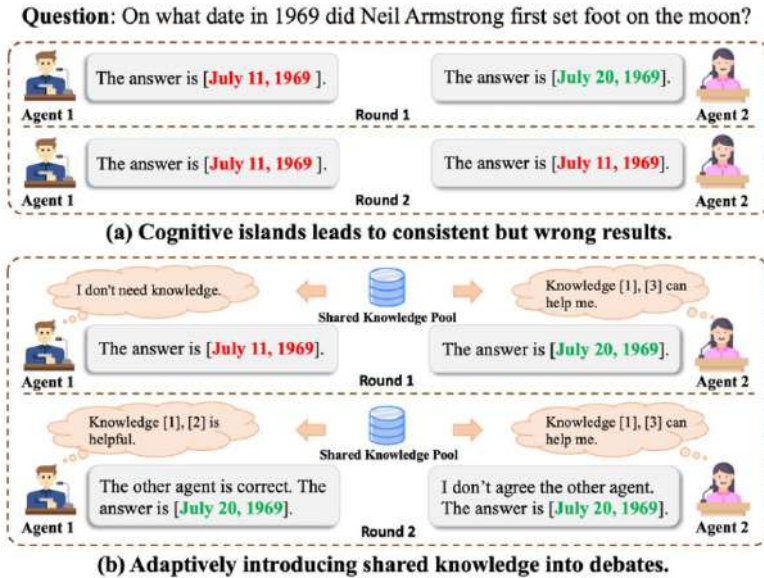


Fig. 1. Example of multi-agent debate scenario with and without the knowledge base [6].

Although the methods mentioned above help reduce hallucinations in LLM agents, the risk is not eliminated. In MAS, this issue is particularly critical, as most tasks require reasoning, and LLMs may struggle to provide coherent answers even for simple tasks [7]. For this reason, we introduce another method – an approach that has long been used prior to recent advancements in AI: logical reasoning.

2.2 Logical Reasoning

In addition to existing machine learning approaches, there is an alternative method in which LLMs are excluded from the reasoning process altogether to avoid hallucinations. Instead, reasoning is performed by matching goals with facts and rules in a structured knowledge base. This is achieved through logic-based agents, which use ontologies and apply the resolution method for logical inference. The knowledge is structured in the form of formal rules and relationships that agents can process to make decisions. This approach demonstrated improved accuracy and performance in question-answering tasks compared to popular LLMs such as GPT-4 [8].

In our current research on developing logic-based agents, we utilize Prolog. Prolog is well-suited for this task because it does not require step-by-step algorithmic instructions for reasoning; instead, it matches facts to rules using first-order logic.

Facts represent known truths. For example, in the case of a MAS for plant care, we can define that a humidity sensor is a sensor, it monitors humidity, and it is associated with a specific plant. This is demonstrated in Listing 1.

```
plant(rose_1).
sensor(humidity_sensor_1).
monitors(humidity_sensor_1, humidity).
connected_to(rose_1, humidity_sensor_1).
```

Listing 1. Examples of facts in Prolog.

Rules are logical statements that describe relationships between facts and allow the inference of new information. For example, we can define a predicate that states that a plant needs watering if the humidity level detected by the sensor connected to it is below the defined threshold (see Listing 2).

```
needs_watering(Plant) :-
    connected_to(Plant, Sensor),
    monitors(Sensor, humidity),
    current_humidity(Sensor, H),
    humidity_threshold(Plant, T),
    H < T.
```

Listing 2. Example of a rule in Prolog.

Prolog-based agents handle interactions from users or other agents through queries – essentially, questions the agent attempts to answer using its existing facts and rules. The reasoning process follows a depth-first search strategy, exploring possible solutions and backtracking when a chosen path does not lead to a valid answer. Failures occur when variable matching is unsuccessful, as Prolog depends on a unification mechanism to align query terms with the knowledge stored in its database.

3. The Proposed Approach

The core idea of our approach is that the impact of hallucinations from LLM agents on a MAS can be almost entirely eliminated by relying on logic-based agents for reasoning. In this setup, the LLM agent primarily functions as a user interface, acting as a bridge between the user and the system.

Given that the agents within the MAS are based on Prolog logic, we propose the possibility of creating an LLM agent capable of generating Prolog queries from user input and directing them to the appropriate agent in the MAS for processing (see Fig. 2).

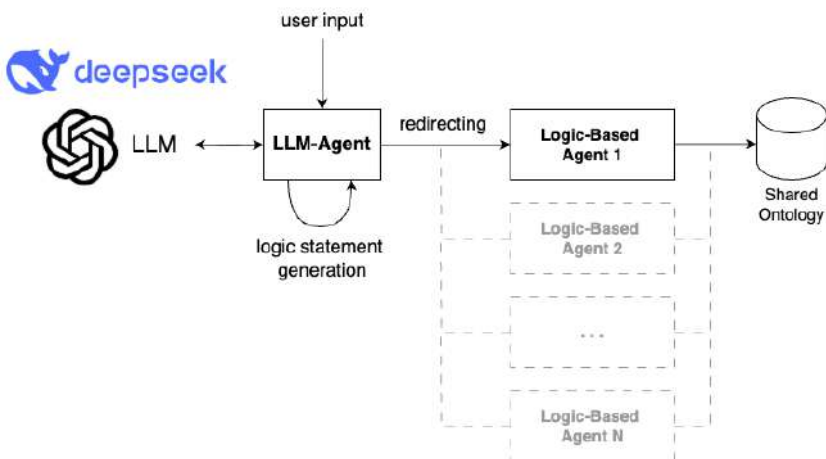


Fig. 2. Generalized representation of the proposed LLM agent workflow.

3.1 Facts and Rules Extraction

Since LLMs have demonstrated strong performance in various text-related tasks, such as semantic parsing and machine translation [9], we decided to employ an LLM-driven parsing approach to transform natural language into Prolog facts and rules. Although it is possible to use an LLM without any additional setup, we considered utilizing in-context learning and CoT to improve accuracy.

The prompt given to the LLM follows a structured format that remains consistent across all queries, to ensure consistency and effectiveness. The prompt consists of the following components:

- *Instruction* – An imperative statement that gives generalized description of the task.
- *Context* – A set of ontology facts and rules that provide the necessary knowledge for accurate translation.
- *Example* – A sample input and output demonstrating a similar task.
- *User Prompt* – An input text string from the user to MAS.

Algorithms that implement LLM-based parsing into logical statements have already been developed [10]. However, these methods share a common limitation: the user must provide the complete context for the LLM to accurately translate the query based on the available knowledge. We propose an alternative that avoids providing the entire ontology to the LLM, as doing so may degrade performance. Instead, we suggest using Retrieval-Augmented Generation (RAG) [11], particularly embeddings, which can effectively represent the ontology. This approach allows us to apply any ontology to our method, as popular ontology formats and Prolog can be converted into vector representations [12]. In future work, we plan to explore this by experimenting with different ontologies as knowledge representations for our agents.

3.2 Correction Algorithm

We need to ensure that the logical statements generated by the LLM agent are correct both in terms of meaning and syntax. To achieve this, we have developed a two-step algorithm that first verifies the meaning and then the syntax. To verify the meaning, we utilize the LLM-as-a-judge method, and implicitly previously mentioned multi-agent debate method.

The first step of the algorithm involves two different LLMs: the first is the LLM agent that generates the Prolog query, and the second is a secondary LLM that acts as a judge to determine whether the generation is correct (see Algorithm 1). We can only proceed to the next step once the secondary LLM is satisfied with the generation. If the generation is deemed unsuccessful, the judge produces a response identifying the issue. The LLM agent will then repeat the generation, using both the user prompt and the error message from the judge as input. This process continues until the result is deemed satisfactory by the judge.

It should be noted that there is a possibility of getting stuck in an infinite loop; however, we can limit the number of verification loops to avoid this.

The second step of the algorithm involves syntax checking. The LLM agent can have a built-in Prolog compiler to perform this check, or it can delegate the task to another agent. In our example, the LLM agent utilizes the compiler (see Algorithm 2) and attempts to execute the code. If the code is not executable, we must repeat step 1 of the algorithm.

3.3 Safe Reasoning

Although we suggest that the Prolog statements generated by the LLM agent are syntactically correct and use terms available in the ontology, we cannot guarantee that the LLM agent will not generate statements that are unsafe to execute. To address this, we developed a Safety-Reasoning Agent (SRA). This agent is responsible for maintaining an ontology of safety rules, all of which must be satisfied at any given time within the system. It ensures that the execution of a command or the introduction of a new fact does not violate these safety rules.

Algorithm 1 Generation and correction of Prolog statements

```
1: Input:  $P \leftarrow$  User prompt
2:    $E \leftarrow$  Error (initially NULL)
3:    $Ontology \leftarrow$  Path to ontology (vector/graph DB)
4:    $PT \leftarrow$  Prompt template
5: Output: Status (success/error message), PrologStatement

6: AttemptCounter = 0
7: Error = NULL
8: while AttemptCounter  $\neq$  N do
9:   AttemptCounter++
10:   $OntologyFragments \leftarrow$  RAGSearch( $P$ , Ontology)
11:   $Prompt \leftarrow$  OntologyFragments.InsertInto( $PT$ )
12:   $Statement \leftarrow$  CallLLM( $Prompt$ , Error)
13:  Valid, Error  $\leftarrow$  Validate( $Statement$ ,  $P$ )
14:  if Error  $\neq$  NULL then
15:    Continue to the Next Iteration
16:  end if
17:  return (SuccessStatus, PrologStatement)
18: end while

19: return ErrorStatus
```

Algorithm 2 Prolog Statement Compiler Check

```
1: Input:  $S \leftarrow$  Statement in Prolog
2:    $P \leftarrow$  Initial prompt
3: Output: Status (success/error message), PrologStatement

4: AttemptCounter = 0
5: Error = NULL
6: while AttemptCounter  $\neq$  N do
7:   AttemptCounter++
8:   Error  $\leftarrow$  CompilerCheck( $S$ )
9:   if Error  $\neq$  NULL then
10:     $S \leftarrow$  Algorithm_1( $P$ , Error)
11:   end if
12:   return (SuccessStatus, PrologStatement)
13: end while

14: return ErrorStatus
```

We must ensure that commands are executed only if they are deemed safe. Therefore, before interacting with any agent, the LLM agent must consult the Safety-Reasoning Agent (see Fig. 3). The SRA's role is to check the statement generated by the LLM agent against the safe ontology, which is separate from other MAS ontologies. This is a novel approach to creating a logic-based observer agent for MAS that use LLMs, although it is rooted in earlier ideas [13-14].

To demonstrate this approach and the MAS architecture in action, we developed an example of a plant care subsystem for a smart-home system’s MAS. This is illustrated in Figure 4, which shows several logical agents, an LLM agent that acts as a supervisor, and a safety agent.

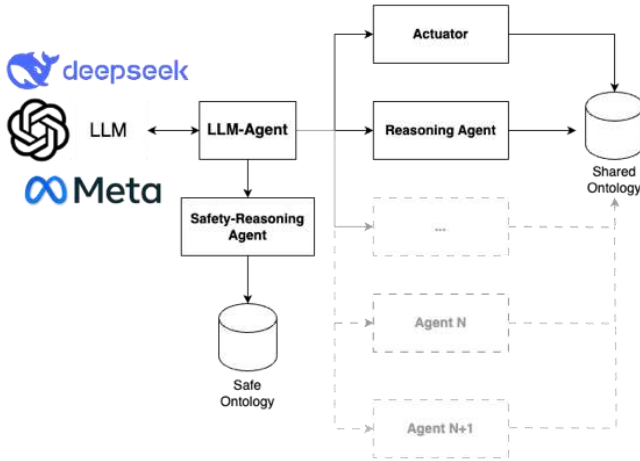


Fig. 3. MAS architecture after the introduction of the safety-reasoning agent

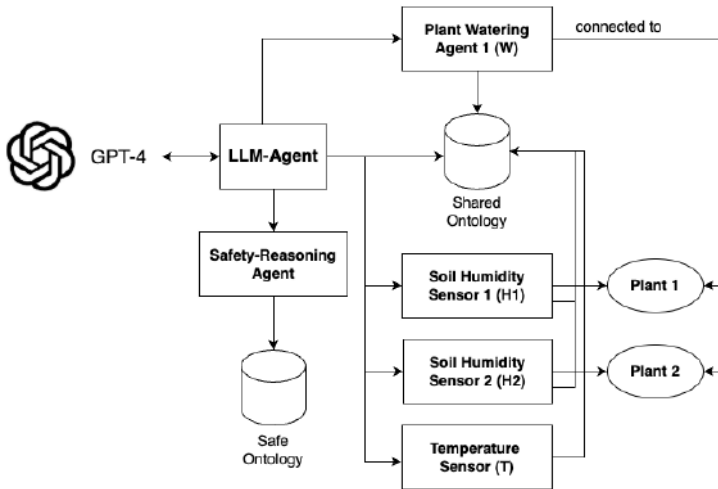


Fig. 4. Example of MAS for plant watering scenario

The process starts with the LLM, which generates commands (or queries) based on user input, and these commands are later executed by MAS agents. The algorithm for generating these commands is outlined in Section 3.2. In our approach, the LLM is supported by an LLM agent, which is tasked with implementing the algorithm. Additionally, the LLM agent functions as a communication facilitator, enabling the exchange of messages (including executable commands) with the appropriate agents.

The remainder of the system can employ actuators or reasoning agents as needed. The MAS benefits from agents built on formal logic models, enabling reasoning without the hallucinations often encountered with LLMs. Agents can either share a common ontology or work with their own individual ones. However, we suggest maintaining a shared ontology for facts while defining rules within each agent separately.

4. Experiments

We have proposed a method for integrating LLM agents with logic-based agents in MAS. The next step is to assess the accuracy of the LLM agent to determine if it introduces hallucinations. We also plan to evaluate the efficiency of the algorithm outlined in Section 3.2. Additionally, we will run the same scenarios with and without the safety-reasoning agent to observe its impact on execution speed. In this chapter, we present a simple MAS example that incorporates an LLM agent designed according to our approach. The system demonstrates a subsystem for a smart home, specifically focusing on plant watering. A schematic representation of the system is shown in Figure 4.

4.1 Overview

The MAS incorporates several Prolog-based actuator agents that share the same ontology and update it when changes occur in the environment:

- Soil Humidity Sensor (H) – Responsible for monitoring and handling changes in soil humidity. There are several agents, each connected with its own plant.
- Temperature sensor (T) – Reacts to changes in the air temperature.

Additionally, there is a reasoning plant-watering agent (W), which manages turning the turning on/off of the watering. To achieve this, it utilizes decision-making: the plant should be watered only when the soil humidity is low, and the temperature exceeds the minimum required for watering.

```
should_water(Plant) :-  
    soil_moisture(Plant, low),  
    temperature(Plant, Temp),  
    min_temp_for_watering(Plant, MinTemp),  
    Temp >= MinTemp.
```

Listing 3. Plant watering rule.

We utilize OpenAI's GPT-4 which is accessed by the LLM agent. When a query is received, the LLM agent relays it to the Safety Reasoning Agent (SRA), which assesses whether the query can be safely executed. In our scenario, activating the watering system is deemed safe as long as the maximum soil humidity level has not been exceeded. If this condition is met, the LLM agent then sends the command directly to the physical watering agent.

4.2 Scenarios Testing

To evaluate the system's performance, we implemented the SRA , W , T , and H agents in Prolog, as described earlier. For the LLM agent, we created a separate Python script that generates a text embedding based on the ontology's classes, relations, and properties. The script processes user input, constructs a prompt, and sends it to the OpenAI API. The Prolog query is then extracted using the algorithms outlined in Section 3.2. This query is executed by the Prolog agents, which return either the result or an error if the query cannot be processed.

We conducted several tests, including:

- 1) Triggering an actuator (H agent, T agent).
- 2) Calling the reasoning W agent to control watering (on/off).
- 3) Retrieving knowledge from the ontology.
- 4) Modifying the ontology.

An example of the test output for the plant watering scenario is provided in Listing 4.

We measured execution times for all scenarios in two variations: (1) when SRA was present in the system, (2) when it was not included in the scenario to see whether it will significantly affect the execution time. The results are presented in Tables 1 and 2.

```

User Input:
"Water the ficus, please"
Attempt 1: Generated Prolog Query:
execute_watering(ficus).
Generation time: 0.7765 seconds
Validation time: 3.1058 seconds
Query is valid, sending to Prolog server
Response from Prolog server:
{
  'status': 'success',
  'result': 'watering_module_1
            is watering ficus.\n
            Moisture is now medium
            for ficus.\n'
}
Prolog execution time: 0.5929 seconds
Total execution time: 4.4753 seconds
    
```

Listing 4. Example of an output of the testing program.

Table 1. Test of Query Generation.

| Scenario № | Generation Time, s | Validation Time, s |
|------------|--------------------|--------------------|
| 1 | 1.7311 | 9.1456 |
| 2 | 0.7765 | 3.1058 |
| 3 | 1.5481 | 7.2752 |
| 4 | 1.4584 | 0.6220 |

Table 2. Execution Time Test.

| Scenario № | Without SRA, s | With SRA, s |
|------------|----------------|-------------|
| 1 | 0.5919 | 0.5955 |
| 2 | 0.5911 | 0.5929 |
| 3 | 0.5921 | 0.5931 |
| 4 | 0.5913 | 0.5918 |

We observed that the execution time difference with and without the SRA was minimal, the Prolog execution time overall was consistently under 0.6 seconds. Additionally, in this example the system demonstrated 100% accuracy in constructing Prolog queries, effectively mitigating hallucination issues. However, this comes at a significant time cost: while generating the initial query takes about 1 second (thanks to embeddings), the evaluation process can take over 9 seconds, which depends on the complexity and length of the query.

4.3 Simulations Comparison

After implementing the test version of the system and conducting the initial evaluation, the following questions also needed to be answered:

1. Does the proposed approach really reduce the effect of hallucinations on MAS?
2. Is it possible to block the hallucinatory actions of the LLM agent?

3. Will it be possible to compensate for the small number of LLM model parameters by using the proposed approach?

To answer each of these questions, two systems (with and without using the proposed approach) were developed to model a MAS that simulates a greenhouse, its environment and the system controlling its agents.

System #1 (see Fig. 5) does not utilize the approach developed in the study. It includes sensor agents:

- Soil Humidity Sensor – an agent responsible for monitoring changes in soil moisture. Several such agents may be present in the system, each associated with a different plant.
- Soil Temperature Sensor – an agent responsible for monitoring changes in soil temperature. Several such agents may be provided in the system, each associated with a different plant.
- Air Temperature Sensor – an agent responsible for responding to changes in air temperature.
- Air Humidity Sensor – an agent responsible for monitoring changes in air humidity.
- Carbon Dioxide (CO2) Sensor – an agent responsible for monitoring changes in the level of carbon dioxide in the air (percentage).

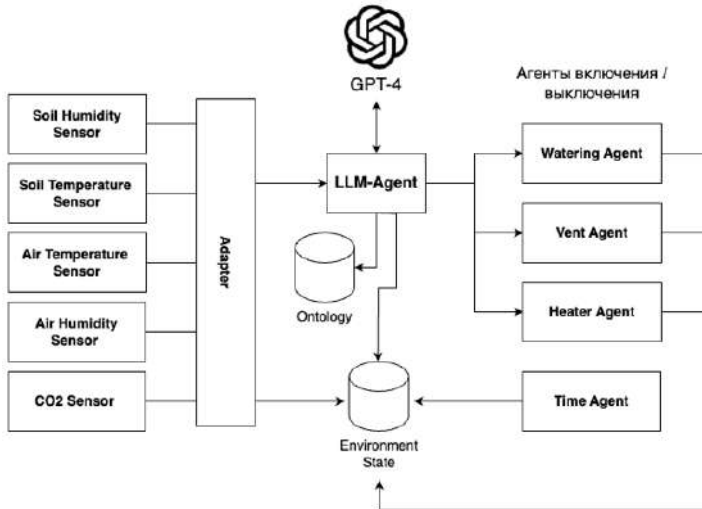


Fig. 5. Representation of a greenhouse simulation. Suggested approach is not implemented.

The state of the greenhouse environment is simulated by sensor agents, as well as a time agent that updates the time within the system, the time of day in the simulation (morning, day, evening, night). The indicators of the external environment (humidity, temperature) can be influenced by action agents:

- Heater – an agent responsible for turning the heating on and off. It simulates power consumption and updates the air temperature value.
- Watering – an agent responsible for turning watering on and off. It simulates water consumption and updates the soil humidity value.
- Vent – an agent responsible for turning ventilation on and off. Simulates electricity consumption and updates the value of carbon dioxide level in the air.

These agents switch between two states – activation and deactivation. In active mode, the agents change the value of a certain environmental parameter at regular intervals (set within the agent) until they are stopped. To control the action agents, an LLM agent is included in the system. Its task is to

decide on the activation/deactivation of the action agent depending on the state of the environment broadcast by the sensor agents. Thus, in this case, the agents of the system do not reason, but only perform actions at the request of the LLM agent.

Logging to .csv files was implemented to track changes within the system. Each record contains information necessary for analysis (see Table 3), as well as for working with process mining tools (such as ProM or pm4py). One log file record contains information about the day number within the simulation (case_id), time, environment parameters (air temperature and humidity, soil temperature, etc.), amount of consumed water and electricity, the event that occurred, and which agent caused it. System #2 (see Fig. 6) utilizes the approach developed in the study. The MAS includes sensor agents similar to System #1, as well as an LLM agent that redirects requests from the sensor agents to the action agents. However, in this case, the LLM agent is not responsible for making the final decision as to which agent will be activated. Action agents in this case are Prolog-agents that check the state of the system regarding facts that will result in an agent not being switched without explicitly stating so in the ontology. For example, a plant watering agent that controls the on/off of watering uses the rule from Listing 3: a plant should only be watered when the soil moisture is low, and the temperature is above the minimum required for watering.

The GPT-4 model from OpenAI was used in the MAS implementation, access to which was granted to the LLM agent. It should be noted that the LLM agent was implemented similarly to the one presented in Section 4.1.

The difference between the LLM agent of System #2 and System #1 is also that when a response with a command is received from the LLM, the LLM agent passes it to the Safety Reasoning Agent (SRA), which performs a check whether the request can be safely executed. In our scenario, activation of the irrigation system is considered safe as long as the maximum soil moisture level is not exceeded. If this condition is met, the LLM agent sends the command directly to the logical irrigation agent. It is worth noting that both System #1 and System #2 use the same ontology, i.e., the agent behavior constraints are defined in the same way.

Table 3. Example of a fragment of the simulation system log file.

| case_id | time | day | air_t | air_hum | co2 | soil_t | soil_h | electr | water | triggered_by | event_type | timestamp |
|---------|-------|---------------|-------|---------|-----|--------|--------|--------|----------|-------------------|-------------------------------|---------------------|
| 1 | 00:04 | LATE_NIGHT | 12 | 42 | 30 | 18 | 40 | 0.0 | 0.0 | air_temperature_1 | SENSOR_AIR_TEMPERATURE_stable | 23.05.2025 01:19:29 |
| 1 | 04:46 | EARLY_MORNING | 20 | 52 | 26 | 12 | 93 | 7000 | 4248.3 | ACTOR_VENT | ACTOR_VENT_active | 23.05.2025 01:24:11 |
| 1 | 04:53 | EARLY_MORNING | 14 | 53 | 26 | 8 | 94 | 7000 | 4414.9 | ACTOR_WATERING | ACTOR_WATERING_active | 23.05.2025 01:24:18 |
| 2 | 10:41 | MORNING | 19 | 46 | 10 | 17 | 97 | 49250 | 32820.19 | soil_humidity_1 | SENSOR_SOIL_HUMIDITY_decrease | 23.05.2025 01:54:14 |
| 2 | 10:42 | MORNING | 25 | 46 | 10 | 21 | 92 | 49500 | 32820.19 | ACTOR_HEATER | ACTOR_HEATER_active | 23.05.2025 01:54:15 |
| 2 | 10:44 | MORNING | 25 | 46 | 10 | 19 | 100 | 49500 | 32903.49 | ACTOR_WATERING | ACTOR_WATERING_active | 23.05.2025 01:54:17 |
| 3 | 00:38 | LATE_NIGHT | 17 | 57 | 42 | 16 | 97 | 70625 | 43565.9 | carbon_1 | SENSOR_CO2_increase | 23.05.2025 02:08:16 |

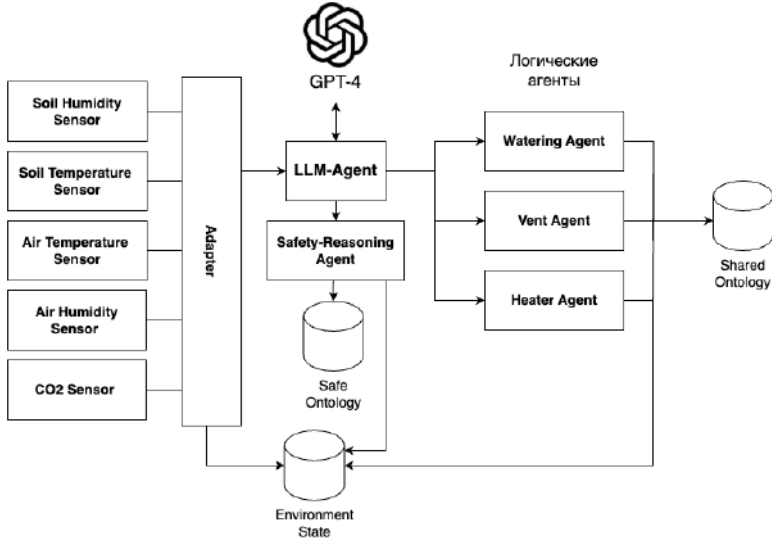


Fig. 6. Representation of a greenhouse simulation. Suggested approach is implemented.

As part of testing, the implemented MASs were run, simulating changes in the external environment and greenhouse agents. The log files were collected and analyzed. As a result, the question of reducing the effect of hallucinations on MAS can be looked at from different perspectives:

- From the point of view of occurrence of undesirable actions – in this case it is necessary to check that agents in both systems do not allow dangerous actions (such actions are marked in the ontology).
- From the point of view of energy efficiency of the whole system – in this case it is necessary to consider what impact the system has on water and electricity consumption.

As a result of analyzing the log files, it was revealed that System #1 allowed actions that can be characterized as “dangerous”. Also, a number of actions that were inefficient in terms of resource consumption were identified (for example, switching on the ventilation and irrigation agent occurred when the values were within the acceptable level). System #2 allowed a different number of “dangerous” actions in each test run, but this number was lower than for System #1. Also, during the testing of System #2 no actions of the agents that were not justified by the environmental factors were observed.

Both systems were also tested with different variations of LLMs connected to the agent LLM (variations of gpt-3 and gpt-4). Regardless of the LLM configuration, System #2 proved to be more energy efficient and safer compared to the system without logical agents.

5. Developing the Hardware

Currently, we are exploring the possibility of creating a real-life prototype of a smart home system based on our approach. To facilitate this, we have developed a watering device. The hardware subsystem is built on the Raspberry Pi Pico W board, which features the RP2040 microcontroller and provides network interaction with the server part via the built-in Wi-Fi module. Figure 7 shows a schematic diagram of the device, which includes the following components:

- Raspberry Pi Pico W microcontroller – the central component of the system, responsible for data processing, controlling connected devices, and networking.
- Servo actuator – controls the water supply process.
- Soil moisture sensor – provides data for analyzing the condition of the crop environment.

- LED – serves as a visual indicator of the device's status.
- Power supply – ensures the stable operation of the device.

The idea is for soil moisture sensor to continuously check the moisture content of the soil, sending the data to the server. When the moisture level falls below the preset threshold, the LLM agent should trigger the watering process, activating the servo actuator to release water to the plants. Afterward, the device checks the moisture level again to determine if the required watering amount has been reached.

Currently, the device allows users to remotely monitor and control the watering system, providing an interface for manual intervention if needed. In future iterations, we plan to enhance the system's capabilities by incorporating more advanced features, such as adaptive watering schedules based on weather forecasts or plant types, integrating additional sensors.

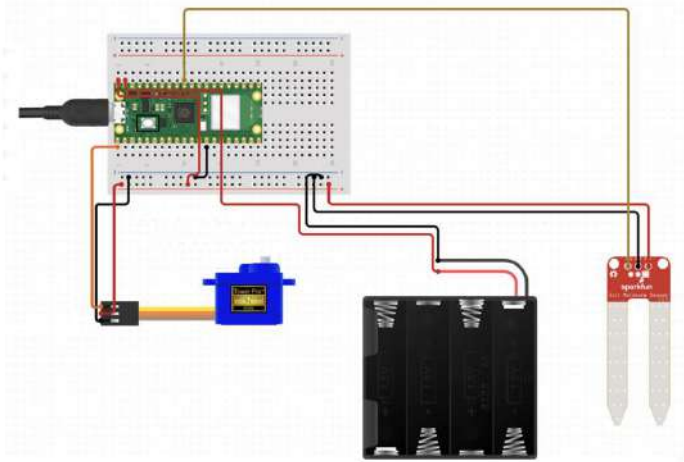


Fig. 7. The schema of plant watering device.

6. Related Works

This paper investigates the potential to fully eliminate the risk of hallucinations. To achieve this, reasoning in the agents is carried out by aligning their goals with facts and rules from a structured knowledge base. This method has proven effective and has been successfully used in the development of chatbots to enhance creativity and maintain focus on specific topics [8]. The combination of logic and ontologies has also been shown to improve accuracy and efficiency in tasks like question answering, outperforming popular LLMs [15]. While this paper focuses on Prolog ontologies and Prolog-based agents, it's important to note that the same approach can be implemented using other declarative programming languages, such as ASP (Answer Set Programming) [16].

Though the combination of LLMs and logic is a well-explored area with various applications [17-18] limited research has been conducted in the context of multi-agent systems (MAS). Our work places particular emphasis on ensuring system safety by introducing a safety-reasoning agent. A similar concept has been explored in previous studies [13-14], showing that although these works are not recent, their foundational ideas can still be adapted and applied today.

7. Future Work

In future research, we plan to explore additional MAS architectures and configurations to broaden the applicability of our approach. We will also investigate the algorithms for the dynamic enrichment of ontologies, allowing for more adaptive knowledge representation. As part of our ongoing work,

we intend to further develop the smart-home system based on the principles outlined in this paper. This will involve refining and optimizing the algorithms we've designed to improve their performance and scalability, ensuring that our solution can handle more complex scenarios and larger-scale implementations. Additionally, there is a big potential to explore new techniques such as real-time learning and cross-agent collaboration, to enhance the system's capabilities and efficiency.

8. Conclusion

In this paper, we introduced a novel approach for integrating Large Language Models (LLMs) into Multi-Agent Systems (MAS) with a focus on minimizing the risks of hallucinations and enhancing system safety. Our method leverages LLMs primarily for the transformation of natural language into logical statements, while the decision-making process is carried out by predicate logic agents. To ensure the correctness of the generated logical statements, we developed a two-step algorithm that incorporates meaning verification and syntax checking. Additionally, we introduced a Safety-Reasoning Agent (SRA), which ensures that only safe actions are executed within the system.

Our results, demonstrated in a smart home automation scenario focused on plant care. The developed two-step algorithm for generating Prolog logic statements from natural language proved successful in reducing hallucinations and ensuring the correctness of the generated code. The incorporation of the Safety-Reasoning Agent strengthened the system by preventing unsafe actions. Despite the promising results, the system's validation process remains computationally intensive, suggesting a need for further optimization in future work in terms of scalability.

We believe that this approach holds significant potential for application in other Multi-Agent Systems, particularly in the context of smart homes, where the integration of LLMs can improve user experience. The inclusion of a Safety-Reasoning Agent introduces an additional layer of reliability, which is crucial for environments where human safety and security are of the primary importance.

References

- [1]. Janbi N., Katib I., Mehmood R. Distributed artificial intelligence: Taxonomy, review, framework, and reference architecture. *Intelligent Systems with Applications*, vol. 18, 2023, p. 200231. DOI: 10.1016/j.iswa.2023.200231.
- [2]. Guo T., Chen X., Wang Y., Chang R., Pei S., Chawla N.V., Wiest O., Zhang X.N. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint*, 2024. DOI: 10.24963/ijcai.2024/890.
- [3]. Christman J., Mele A.R. Autonomous agents: From self-control to autonomy. *The Journal of Philosophy*, vol. 96, no. 2, 1999, pp. 95–100. DOI: 10.2307/2564674.
- [4]. Wei J., Wang X., Schuurmans D., Bosma M., Ichter B., Xia F., Chi E., Le Q.V., Zhou D. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint*, vol. arXiv:2201.11903, 2023. Available at: <https://arxiv.org/abs/2201.11903>.
- [5]. Wang H., Du X., Yu W., Chen Q., Zhu K., Chu Z., Yan L., Guan Y. Learning to break: Knowledge-enhanced reasoning in multi-agent debate system. *Neurocomputing*, vol. 618, 2025, p. 129063. Available at: <https://www.sciencedirect.com/science/article/pii/S0925231224018344>.
- [6]. Jiang Y.-H., Li R., Zhou Y., Qi C., Hu H., Wei et al. AI agent for education: von Neumann multi-agent system framework. In *Proc. of the 28th Global Chinese Conference on Computers in Education*, 2024.
- [7]. Nezhurina M., Cipolina-Kun L., Cherti M., Jitsev J. Alice in Wonderland: Simple tasks showing complete reasoning breakdown in state-of-the-art large language models. *arXiv preprint*, vol. arXiv:2406.02061, 2024. Available at: <https://arxiv.org/abs/2406.02061>.
- [8]. Zeng Y., Rajasekharan A., Basu K., Wang H., Arias J., Gupta G. A reliable common-sense reasoning socialbot built using LLMs and goal-directed ASP. *Theory and Practice of Logic Programming*, vol. 24, no. 4, 2024. DOI: 10.1017/S147106842400022X.
- [9]. OpenAI. GPT-4 Technical Report. *arXiv preprint*, vol. arXiv:2303.08774, 2024. Available at: <https://arxiv.org/abs/2303.08774>.

- [10]. Wang Z., Liu J., Bao Q., Rong H., Zhang J. ChatLogic: Integrating logic programming with large language models for multi-step reasoning. arXiv preprint, vol. arXiv:2407.10162, 2024. Available at: <https://arxiv.org/abs/2407.10162>.
- [11]. Lewis P.S.H., Perez E., Piktus A., Petroni F., Karpukhin V., Goyal N., Kuttler H., Lewis M., Yih W.-T., Rocktäschel T., Riedel S., Kiela D. Retrieval-augmented generation for knowledge-intensive NLP tasks. arXiv preprint, vol. arXiv:2005.11401, 2020. Available at: <https://arxiv.org/abs/2005.11401>.
- [12]. Chen J., Hu P., Jiménez-Ruiz E., Holter O.M., Antonyrajah D., Horrocks I. OWL2Vec*: Embedding of OWL ontologies. arXiv preprint, vol. arXiv:2009.14654, 2020. Available at: <https://arxiv.org/abs/2009.14654>.
- [13]. Modgil S., Fox J. A guardian agent approach to safety in medical multi-agent systems. In *Safety and Security in Multiagent Systems*, 2009.
- [14]. Nimis J., Lockemann P.C. Robust multi-agent systems: The transactional conversation approach? 2004.
- [15]. Cabalar P., Fabiano F., Gebser M., Gupta G., Swift T. Proceedings of the 40th International Conference on Logic Programming (ICLP 2024). In *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, University of Texas at Dallas, Dallas, TX, USA, Oct. 2024, pp. 69–77.
- [16]. Lifschitz V. What is Answer Set Programming? In *Proceedings of the National Conference on Artificial Intelligence*, vol. 3, 2008, pp. 1594–1597.
- [17]. Lin X., Wu Y.-C., Yang H., Zhang Y., Zhang Y., Ji J. CLMASP: Coupling large language models with answer set programming for robotic task planning. arXiv preprint, vol. arXiv:2406.03367, 2024. Available at: <https://arxiv.org/abs/2406.03367>.
- [18]. Yang Z., Ishay A., Lee J. Coupling large language models with logic programming for robust and general reasoning from text. arXiv preprint, vol. arXiv:2307.07696, 2023. Available at: <https://arxiv.org/abs/2307.07696>.

Информация об авторах / Information about authors

Людмила Александровна РЕЗУНИК – магистр программной инженерии, НИУ ВШЭ. Сфера научных интересов: разработка мобильных приложений, мультиагентные системы, большие языковые модели LLM, архитектура программного обеспечения.

Lyudmila Aleksandrovna REZUNIK – Master of Software Engineering, HSE. Research interests: mobile application development, multi-agent systems, LLM, software architecture.

Михаил Алексеевич ПРОЗОРСКИЙ – студент НИУ ВШЭ, стажер-исследователь научно-учебной лаборатории облачных и мобильных технологий НИУ ВШЭ. Сфера научных интересов: разработка мобильных приложений, системы умного дома, архитектура программного обеспечения.

Mikhail Aleekseevich PROZORSKIY – student at HSE, researcher at the Educational and Research Laboratory of Cloud and Mobile Technologies, HSE. Research interests: mobile application development, smart home systems, software architecture.

Дмитрий Владимирович АЛЕКСАНДРОВ – доктор технических наук, профессор, заведующий научно-учебной лаборатории облачных и мобильных технологий НИУ ВШЭ. Сфера научных интересов: методы и технологии искусственного интеллекта, разработка мобильных приложений, разработка программного обеспечения, инженерия знаний.

Dmitry Vladimirovich ALEXANDROV – Dr. Sci. (Tech.), Professor, Head at the Educational and Research Laboratory of Cloud and Mobile Technologies, HSE. Research interests: methods and techniques of artificial intelligence, mobile application development, software development, knowledge engineering.

DOI: 10.15514/ISPRAS-2025-37(4)-29



Применение динамической символьной интерпретации в гибридном фаззинге бинарного кода для архитектур Байкал-М и RISC-V 64

В. И. Логунова, ORCID: 0000-0002-3877-1906 <vlada@ispras.ru>

*Институт системного программирования им. В.П. Иванникова РАН,
Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.*

Аннотация. Гибридный фаззинг и динамическая символьная интерпретация активно применяются в жизненном цикле разработки безопасного программного обеспечения. В настоящее время доля кода программ, разрабатываемых для архитектур ARM и RISC-V, постоянно увеличивается вместе с потребностью в их эффективном анализе. Данная работа посвящена решению этой задачи путем разработки методов динамической символьной интерпретации и гибридного фаззинга для современных RISC-архитектур – отечественной «Байкал-М» (ARM/AArch64) и открытой RISC-V 64. Разработанные подходы, основанные на моделировании символьной семантики машинных инструкций, интегрированы в инструмент Sydr в составе фреймворка Sydr-Fuzz и нацелены на повышение эффективности гибридного фаззинга. Ключевые результаты включают алгоритмы обработки косвенных переходов для точного определения целевых адресов и реализацию поддержки набора целочисленных инструкций RISC-V в открытом фреймворке Triton, что предоставляет сообществу готовую основу для создания инструментов динамического анализа.

Ключевые слова: динамический анализ; динамическая символьная интерпретация; фаззинг; гибридный фаззинг; инструментация; анализ бинарного кода.

Для цитирования: Логунова В. И. Применение динамической символьной интерпретации в гибридном фаззинге бинарного кода для архитектур Байкал-М и RISC-V 64. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 235–250. DOI: 10.15514/ISPRAS–2025–37(4)–29.

Application of Dynamic Symbolic Execution in Hybrid Fuzzing of Binary Code for Baikal-M and RISC-V 64 Architectures

V. I. Logunova, ORCID: 0000-0002-3877-1906 <vlada@ispras.ru>

*Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*

Abstract. Hybrid fuzzing and dynamic symbolic execution have become a vital part of the secure software development lifecycle. Currently, the proportion of code being developed for ARM and RISC-V architectures is constantly increasing, making the task of their effective analysis a top priority. This work is dedicated to solving this task by developing methods for dynamic symbolic execution and hybrid fuzzing for modern RISC architectures – «Baikal-M» (ARM/AArch64) and RISC-V 64. Based on modeling symbolic semantics of machine instructions the developed approaches are integrated into the Sydr tool within the Sydr-Fuzz framework and aim to enhance the efficiency of hybrid fuzzing. Key results include algorithms for processing indirect branches with accurate target addresses determination and RISC-V integer instruction set support in open-source symbolic framework Triton that provides the community with a ready-made foundation for creating dynamic analysis tools.

Keywords: dynamic analysis; dynamic symbolic execution; fuzzing; hybrid fuzzing; instrumentation; binary code analysis.

For citation: Logunova V. I. Application of Dynamic Symbolic Execution in Hybrid Fuzzing of Binary Code for Baikal-M and RISC-V 64 Architectures. *Trudy ISP RAN/Proc. ISP RAS*, vol. 37, issue 4, part 2, 2025, pp. 235-250 (in Russian). DOI: 10.15514/ISPRAS-2025-37(4)-29.

1. Введение

Программное обеспечение глубоко интегрировано в современную жизнь, от критической инфраструктуры до персональных гаджетов. Эта повсеместность, однако, сопряжена с риском, поскольку в процессе разработки неизбежно возникают программные дефекты в виде ошибок и уязвимостей. Поэтому в качестве стандарта для их детектирования и устранения используется жизненный цикл разработки безопасного программного обеспечения (ПО), включающий различные виды анализа программ [1]. Соответственно, одной из актуальных задач становится разработка универсальных автоматизированных инструментов, способных эффективно обнаруживать ошибки в коде.

Отличительной чертой динамического анализа является запуск исследуемой программы, за счёт чего снижается количество ложных сообщений об ошибках. Классическим методом в данном направлении, на текущий момент, является фаззинг-тестирование с обратной связью по покрытию (фаззинг "методом серого ящика") [2-3]. Ключевую роль для этого вида анализа играет метрика покрытия, достигнутого во время множественных запусков приложения с разнообразными входными данными. Генерация новых входных данных производится с помощью применения случайных мутаций к данным из предыдущих запусков, которые приоритизируются по приросту покрытия. Кроме того, существует фаззинг "методом белого ящика" [4] также известный как динамическая символьная интерпретация. Вместо получения метрики покрытия данный подход извлекает более подробную информацию о внутреннем устройстве программы для последующей генерации входных данных на основе аналитических вычислений. В первую очередь, это логические условия ветвления, зависящие от изначальных помеченных данных. Из таких условий составляется абстрактная (символьная) модель – предикат пути, отражающий зависимости потока управления от входных данных. Новые входные данные подбираются так, чтобы изменить выбор направления ветвления при будущем запуске. Чем больше информации о содержании кода извлекается при анализе, тем более точно можно указать на ошибку и тем больше возможных особенностей кода требуется учесть при создании анализатора для правильной

инструментации. К таким особенностям может относиться язык программирования в случае инструментации исходного кода или архитектура процессора при исследовании бинарного кода. Символьная интерпретация требует больше времени на один запуск по сравнению с обычным фаззингом. Несмотря на это, при совместной работе своевременный обмен входными данными между инструментами способен повышать общую эффективность анализа [5]. Данный подход называется гибридным фаззингом.

2. Создание абстрактной модели программы в динамической символьной интерпретации

В данной работе исследуется применение гибридного фаззинга для анализа бинарного кода архитектур Байкал-М и RISC-V. Как можно заметить, динамическая символьная интерпретация представляет собой достаточно низкоуровневый подход. Одним из важных аспектов является переносимость, то есть возможность анализа программ, разработанных для различных аппаратных платформ. Инструменты отличаются по требованию к наличию исходного кода и выбору представления, для которого формулируются символьные преобразования входных данных. Помимо этого, существуют различные варианты применения конкретного запуска в процессе анализа.

При наличии исходного кода возможна компиляция в универсальное промежуточное представление, при создании которого архитектурно-зависимые компоненты будут оставлены за скобками за счет специфически модифицированного компилятора. Похожий подход применим также непосредственно к бинарному коду, что более трудоёмко из-за частичных потерь информации на этапе компиляции. Однако доступ к исходному коду программы есть не всегда, как, например, в случае использования сторонних динамических библиотек. Кроме того, интеграция анализатора, требующего доступ к исходному коду, предполагает внесение изменений либо непосредственно в код, либо затрагивает технологический стек в виде компилятора и систем сборки ПО, что также вносит ограничения на удобство и универсальность применения инструмента. Таким образом, в контексте динамической символьной интерпретации требование к наличию исходного кода выглядит избыточным, поэтому далее главным образом будут рассмотрены методы инструментации, применимые на уровне бинарного кода.

Наиболее распространенным подходом к динамической символьной интерпретации является так называемое конкретно-символьное исполнение, когда непосредственный запуск исследуемой программы комбинируется с символьной интерпретацией для получения более точной абстрактной модели исполнения программы. Первым инструментом с реализацией такого подхода для бинарного кода стал SAGE [4], где сначала производится конкретный запуск программы для записи трассы исполнения, а затем её преобразование в символьные ограничения. В современных инструментах конкретно-символьного исполнения эти два этапа совмещаются с помощью динамической инструментации, которая предоставляет текущий контекст конкретного исполнения программы (исполняемые инструкции, значения памяти, регистров и флагов) для символьной интерпретации. Например, QSYM [6] и Sydr [7] используют фреймворки динамической бинарной инструментации и выполняют символьную интерпретацию на уровне отдельных инструкций бинарного кода. Для них поддержка каждой новой платформы требует добавления символьного представления для анализируемого набора инструкций выбранной архитектуры. На уровне машинных инструкций, но без реального запуска проходит символьная интерпретация в TritonDSE [8] – инструмент эмулирует бинарный код для пошагового выполнения. При этом, запуск может присутствовать для верификации или сбора покрытия, как и в виртуальной символьной машине KLEE [9], эмулирующей инструкции байт-кода LLVM. Преимущество инструментации промежуточного представления также используется в SymQEMU [10], где перед запуском бинарный код транслируется с помощью QEMU в TCG ops и обратно.

Символьная инструментация производится на уровне этого представления, позволяя абстрагировать инструмент от деталей реализации отдельных аппаратных платформ.

Данная работа по применению символьного анализа бинарного кода архитектур Байкал-М (ARM/AArch64) и RISC-V в гибридном фаззинге выполнялась на основе динамического символьного интерпретатора Sydr, разработанного в ИСП РАН. Инструмент Sydr реализует конкретно-символьное исполнение бинарного кода архитектуры x86/x86-64 с использованием динамического бинарного инструментатора DynamoRIO и библиотеки символьного исполнения Triton.

3. Символьная интерпретация бинарного кода архитектуры Байкал-М (ARM/AArch64)

В предлагаемом методе для инструмента Sydr моделирование символьного контекста выполнения производится для целочисленных инструкций ARM/AArch64. Обработка операций с плавающей точкой при символьной интерпретации программы порождает довольно сложные символьные уравнения, для которых SMT-решатель обычно не дает решение за установленный лимит времени. Соответственно, для архитектуры AArch64 в предлагаемом методе применяется символьная интерпретация на основе следующих компонентов:

- набор регистров общего назначения, включающий регистры x0-x30 размером 64 бита, а также соответствующие им подрегистры w0-w30 размером 32 бита;
- набор специальных регистров, включающих регистр указателя текущей инструкции (pc), регистр стекового указателя (sp), регистр флагов, а также нулевые регистры xzr и wzr;
- операционная семантика инструкций с целочисленными операндами;
- механизмы вычисления адресных выражений;
- механизмы передачи управления и соглашения о вызовах.

Обновление символьного контекста происходит по мере обработки потока инструкций, выполняющихся под контролем DynamoRIO. Получив от инструментатора текущую инструкцию вместе с конкретными значениями, процесс обработки символьного представления дизассемблирует её операционный код и производит набор проверок на наличие символьных операндов и необходимость обновления символьного стека вызовов. Для AArch64 инструкциями вызова функций являются BL и BLR, а для возврата из функции используется RET, т.е. инструкция BR x30, которая передает управление на инструкцию, адрес которой хранится в регистре x30 (Procedure Link Register). Отслеживание данных инструкций необходимо для актуализации символьной модели памяти стека. Иначе при последующих операциях чтения или сохранения символьных данных в стековой памяти интерпретатор обнаружит несоответствие и подставит значение конкретных данных, то есть символьная пометка будет потеряна.

Для архитектуры ARM/AArch64 инструкции обращения к памяти (load/store) выделяются в отдельную группу. Размеры области памяти, к которой может обращаться инструкция, определяются семантикой и составляют 1, 2, 4 или 8 байтов. При реализации предлагаемого метода в абстрактном интерфейсе MemoryAccess библиотеки Triton была исправлена ошибка присвоения значения области памяти размером, тождественным размеру регистра для её временного хранения (4 либо 8 байтов). Среди прочего, для механизма адресации с индексированием в ARM/AArch64 присутствует несколько необычный формат применения коэффициента масштабирования к индексному регистру. Обычно данный коэффициент представляется собой константный числовой множитель. Например, 8 в выражении «Addr = Reg1 + Reg2 * 8». В AArch64 вместо него предусмотрена возможность применения операции

битового сдвига влево (LSL), которая может быть совмещена с операцией расширения индексного регистра (см. рис. 1).

Операция сдвига эквивалентна, по сути, умножению на степени двойки от 0 до 3, позволяя адресовать блоки памяти размером от 1 до 8 байтов. Она также может участвовать в адресации косвенных табличных переходов [11], в том числе, в контексте арифметической операции (например, инструкции сложения из последней строки на рис. 1), выполняемой уже после загрузки значения из памяти. Предлагаемый метод находит такие арифметические преобразования между инструкцией чтения из памяти и передачи управления с помощью алгоритма отслеживания регистров (слайсинга), на основе которого производится коррекция адресного выражения с учетом возможности битовых сдвигов. Само адресное выражение ветвления с несколькими вариантами перехода добавляется в предикат пути позднее, при появлении соответствующей инструкции передачи управления.

| Инструкция | Выполняемая операция |
|--|--|
| <code>ldr x0, [x1, x2, lsl 3]</code> | <code>x0 := Mem[x1 + x2 * 8]</code> |
| <code>ldr x0, [x1, w2, uxtx 3]</code> | <code>x0 := Mem[x1 + ext(w2) * 8]</code> |
| <code>ldrb w0, [x1, w2, uxtx 3]</code> | <code>w0 := ext(Mem[x1 + x2 * 8])</code> |
| <code>add x0, x1, x2, lsl 3</code> | <code>x0 := x1 + x2 * 8</code> |

Рис. 1. Примеры применения битовых сдвигов к операндам инструкций AArch64.

Условные обозначения: := – операция присваивания значения, ext – операция расширения размера, Mem[addr] – значение памяти по адресу addr.

При конструировании символических предикатов безопасности [12], проверяющих возможность ошибки целочисленного переполнения, учитываются как знаковые, так и беззнаковые переполнения для инструкций сложения, вычитания, умножения, а также битовых сдвигов. Для этого производится развертывание АСТ-выражения инструкции, и из него извлекаются подвыражения, соответствующие операндам инструкции. К примеру, выражение инструкции вычитания с битом переноса содержит две операции сложения битовых векторов, где значение исходного операнда вычитаемого будет взято с отрицанием в виде дополнительного узла в дереве. Поскольку инструкция фактически включает две арифметические операции, допускающие возможность переполнения, будет сконструировано два предиката безопасности.

3.1 Метод обнаружения косвенных переходов в бинарном коде

Ветвления с множественным выбором (в виде конструкции switch) предполагают наличие нескольких вариантов выбора цели передачи управления. При использовании оптимизаций компилятора такие переходы производятся с помощью организации целевых вариантов исполняемого кода в виде таблицы, состоящей из последовательно расположенных базовых блоков, соответствующих различным направлениям ветвления. Целевые адреса из такой таблицы исполняемого кода последовательно хранятся в памяти, а выбор нужной ячейки памяти происходит посредством добавления смещения к базовому адресу, где располагается начало таблицы указателей на исполняемый код. На рис. 2 выделен светлой областью базовый блок, в котором происходит чтение адреса целевого перехода из памяти и передача управления, которая в данном случае представлена вызовом функции и осуществляется с помощью инструкции BLR x8. Как можно заметить, для вычисления базового адреса таблицы здесь используются три различные константы (0x420000, 0x28 и 0x180), а также значение регистра x8, которое перед попаданием в данный базовый блок было прочитано из памяти в инструкции LDRB по адресу 0x4006c0. Поскольку первые восемь регистров x0-x7 в AArch64

используются для хранения аргументов в соответствии с соглашением о вызовах, при взгляде на предыдущую инструкцию можно также догадаться, что в регистре x8 представлено потенциально символическое значение, указатель на адрес которого передается в качестве аргумента функции *main*. Таким образом, в зависимости от данного аргумента будет осуществлен вызов одной из функций *func0-func5*.

```
00000000400664 <func5>:
406664: 90000000      adrp   x0, 400000 <_init-0x4e0>
406668: 911ee000      add    x0, x0, #0x7b8
40666c: 17ffffb9      b      400550 <puts@plt>

00000000400670 <func4>:
406670: 90000000      adrp   x0, 400000 <_init-0x4e0>
406674: 911f0c00      add    x0, x0, #0x7c3
406678: 17ffffb6      b      400550 <puts@plt>

0000000040067c <func3>:
40667c: 90000000      adrp   x0, 400000 <_init-0x4e0>
406680: 911f3800      add    x0, x0, #0x7ce
406684: 17ffffb3      b      400550 <puts@plt>

00000000400688 <func2>:
406688: 90000000      adrp   x0, 400000 <_init-0x4e0>
40668c: 911f6400      add    x0, x0, #0x7d9
406690: 17ffffb0      b      400550 <puts@plt>

00000000400694 <func1>:
406694: 90000000      adrp   x0, 400000 <_init-0x4e0>
406698: 911f9000      add    x0, x0, #0x7e4
40669c: 17ffffad      b      400550 <puts@plt>

000000004006a0 <func0>:
4066a0: 90000000      adrp   x0, 400000 <_init-0x4e0>
4066a4: 911fbc00      add    x0, x0, #0x7ef
4066a8: 17ffffaa      b      400550 <puts@plt>

000000004006ac <main>:
4066ac: a9bf7bfd      stp    x29, x30, [sp, #-16]!
4066b0: 910003fd      mov    x29, sp
4066b4: 7100081f      cmp    w0, #0x2
4066b8: 54000101      b.ne  4006d8 <main+0x2c> // b.any
4066bc: f9400428      ldr    x8, [x1, #8]
4066c0: 39400108      ldrb   w8, [x8]
4066c4: f100bd1f      cmp    x8, #0x2f
4066c8: 54000108      b.hi  4006e8 <main+0x3c> // b.pmore
4066cc: 90000000      adrp   x0, 400000 <_init-0x4e0>
4066d0: 911fe800      add    x0, x0, #0x7fa
4066d4: 14000003      b      4006e0 <main+0x34>
4066d8: 90000000      adrp   x0, 400000 <_init-0x4e0>
4066dc: 91203400      add    x0, x0, #0x80d
4066e0: 97ffff9c      bl     400550 <puts@plt>
4066e4: 14000007      b      400700 <main+0x54>
4066e8: 90000109      adrp   x9, 420000 <__libc_start_main@GLIBC_2.17>
4066ec: 9100a129      add    x9, x9, #0x28
4066f0: 8b080d28      add    x8, x9, x8, lsl #3
4066f4: d1060108      sub    x8, x8, #0x180
4066f8: f9400108      ldr    x8, [x8]
4066fc: d63f0100      blr    x8
406700: 2a1f03e0      mov    w0, wzr
406704: a8c17bfd      ldp    x29, x30, [sp], #16
406708: d65f03c0      ret
```

Рис. 2. Пример организации таблицы переходов в ассемблерном коде.

Детектирование таких таблиц в предлагаемом методе происходит на основе эвристики в виде присутствия в базовом блоке следующих составляющих:

- в качестве последней инструкции используется безусловная инструкция передачи управления по регистровому значению;
- соответствующий регистр хранит значение, прочитанное из памяти, либо является его производным;
- в базовом блоке присутствует одна из инструкций загрузки фиксированного адресного значения ADR/ADRP.

Описанная конструкция может быть дополнительно усложнена, если вместо целевого адреса перехода требуется прочитать из памяти смещение относительно начала таблицы в исполняемом коде. Таким образом, адрес в таблице с исполняемыми базовыми блоками также разбивается на базовый адрес её начала и смещение. В свою очередь, хранение значений отступов аналогично организуется в виде хранящейся в памяти таблицы смещений, для которой осуществляется операция чтения с помощью инструкции загрузки данных (load) тоже на основе индексируемого базового адреса. Размеры блоков памяти для хранения отступов составляют от 1 до 8 байтов. Поскольку ресурс значения смещения для кодирования номера инструкции или записи, отвечающей за конкретный базовый блок, ограничен, при добавлении значения смещения в адресное выражение, оно умножается на соответствующий размер. Как было упомянуто в ранее, в случае архитектуры AArch64 для подобных вычислений используется встроенный механизм арифметических инструкций в виде операции битового сдвига на константное значение.

На рис. 3 представлен базовый блок с передачей управления, основанной на вычислении целевого адреса с помощью отступа, прочитанного из таблицы смещений. Первые две инструкции содержат константы для определения базового адреса таблицы отступов, а третья загружает адрес начала таблицы в исходном коде. Благодаря тому, что присутствие таблицы смещений обуславливает наличие двух базовых адресов вместо одного, можно отличить данный вариант организации таблицы от предыдущего (как на рис. 2) по второй инструкции загрузки адресного значения.

| | | | |
|---------|----------|------|-------------------------------|
| 4006c0: | 54000241 | b.ne | 400708 <main+0x54> // b.any |
| 4006c4: | f9400428 | ldr | x8, [x1, #8] |
| 4006c8: | 39400108 | ldrb | w8, [x8] |
| 4006cc: | 51018908 | sub | w8, w8, #0x62 |
| 4006d0: | 7100211f | cmp | w8, #0x8 |
| 4006d4: | 54000428 | b.hi | 400758 <main+0xa4> // b.pmore |
| 4006d8: | 90000009 | adrp | x9, 400000 <_init-0x520> |
| 4006dc: | 91206129 | add | x9, x9, #0x818 |
| 4006e0: | 1000008a | adr | x10, 4006f0 <main+0x3c> |
| 4006e4: | 3868692b | ldrb | w11, [x9, x8] |
| 4006e8: | 8b0b094a | add | x10, x10, x11, lsl #2 |
| 4006ec: | d61f0140 | br | x10 |
| 4006f0: | 90000000 | adrp | x0, 400000 <_init-0x520> |

Рис. 3. Базовый блок косвенной передачи управления на основе таблицы смещений.

Для составления предиката пути, способного учитывать все доступные варианты, требуется реализовать не только метод распознавания подобных конструкций, но и определения размеров таблицы. Предлагаемый метод определения границ таблицы переходов основан на поиске инструкции сравнения *CMP* в предыдущем логическом базовом блоке. Для этого применяется механизм кэширования выполненных базовых блоков. Путем извлечения константного значения для инструкции сравнения определяются целевые ячейки читаемой памяти. Для их содержимого вычисляются адреса возможных переходов, которые верифицируются на принадлежность к областям исполняемого кода.

4. Символьная интерпретация бинарного кода архитектуры RISC-V

Открытая процессорная архитектура RISC-V появилась сравнительно недавно. Особенности данного архитектурного стандарта являются свободная лицензия, модульность и минималистичность набора инструкций, что делает его применимым для широкого класса приложений.

4.1 Конструирование символьной семантики целочисленных инструкций RISC-V

Предлагаемый метод реализован на базе фреймворка Triton [13]. Метод способен конструировать в терминах битовых векторов символьные AST-выражения, отображающие операционную семантику инструкций RISC-V, для которых осуществляется символьная интерпретация. Соответствующие функциональные методы и генерируемые ими символьные выражения в данной работе также называются "символьной семантикой".

Набор инструкций архитектуры RISC-V имеет модульную структуру из подключаемых расширений, обозначаемых латинскими буквами. К базовому I-расширению относятся инструкции передачи управления, обращений к памяти, основные арифметические операции (кроме принадлежащих M-расширению инструкций умножения и деления). Помимо того, в базовом наборе содержатся инструкции организации прерываний и барьеров памяти (fence-инструкции), а также инструкции для доступа к служебному регистру CSR, но эти группы инструкций не включаются в предлагаемый метод. Значительная часть набора интерпретируемых методом 64-битных инструкций представляет собой пересечение с 32-битной версией архитектуры, отличаясь только размером используемых регистровых операндов и адресов. По данной причине, метод охватывает как набор модульных расширений RV64IMC, так и RV32IMC, где буквой C обозначается набор сокращенных (compressed) инструкций.

Для представления выбранной архитектуры с помощью инфраструктуры абстрактных интерфейсов Triton были добавлены соответствующие спецификации, позволяющие организовывать взаимодействие с внешним дизассемблером Capstone и производить моделирование символьного состояния программы. После создания спецификаций и идентификаторов в предложенном методе были реализованы методы интерфейсов `riscv64Cpu` и `riscv32Cpu`, с помощью которых обрабатываются архитектурно-зависимые компоненты на основе абстрактных универсальных классов (например, `triton::arch::Register`). Данные интерфейсы позволяют получать и обновлять символьные выражения и конкретные значения регистров и областей памяти, осуществлять доступ к регистрам служебного назначения (`sp`, `pc`), а также содержат функции для преобразования контекста дизассемблированных посредством Capstone инструкций в абстрактный класс `triton::arch::Instruction`.

На этапе конструирования символьной семантики с помощью идентификатора инструкции выбирается соответствующий функциональный метод. В первую очередь, он извлекает операнды инструкции и получает для них битовые выражения символьного контекста выполнения. Таким выражением может быть как ранее построенное абстрактное дерево, так и битовый вектор, соответствующий значению константы для заданного размера операнда. Далее для полученных выражений в соответствии с операционной семантикой инструкции строится новое выражение, содержащее их в виде поддеревьев. Сконструированное выражение ассоциируется с целевым операндом, после чего увеличивается символьное значение счётчика инструкций, если иное не предусмотрено семантикой инструкции. Предлагаемый метод обеспечивает для архитектуры RISC-V формирование семантических выражений в терминах битовых векторов для 62 инструкций стандартного размера, их 19 дополнительных вариаций в виде псевдоинструкций, а также 33 сокращенных инструкций.

Псевдоинструкции представляют собой альтернативное прочтение стандартных инструкций при использовании специфических комбинаций операндов. Чаще всего в качестве такого операнда выступает всегда равный нулю регистр $x0$. Также используется регистр $x1$ и константные операнды. Одна и та же инструкция может иметь несколько вариаций в виде псевдоинструкций в зависимости от выбора операндов. Например, инструкция передачи управления `JALR rd, rs, offset` имеет три варианта псевдоинструкций, для каждой из которых значение численного операнда `offset` приравниваются к нулю, в то время как `rd` обязательно принимает значение $x0$ либо $x1$. В случае комбинации `JALR x0, x1, 0` получается инструкция возврата из вызова `RET`. Из-за особенностей работы Capstone, поля предопределяющих семантику операндов не заполняются при дизассемблировании. К примеру, абстрактный класс, отображающий псевдоинструкцию `RET` не имеет ни одного операнда. С точки зрения построения символической семантики, это означает, что перед получением битвекторных выражений необходимо определить, является ли интерпретируемая инструкция стандартной или представляет собой псевдоинструкцию. Для этого в предлагаемом методе применяется разбор на основе количества операндов и текстового описания инструкции.

Сокращенные инструкции образуют набор базовых операций в виде передачи управления, обращений к памяти и простых арифметических операций. Основное их отличие заключается в том, что на их кодирование отводится 16 бит вместо 32, что следует учитывать при обновлении символического значения счетчика инструкций. Кроме того, из-за отсутствия корректной трансляции для инструкций обращения к памяти при разработке предлагаемого метода создание операнда класса `MemoryAccess` для таких инструкций производится в соответствующем функциональном методе для создания символической семантики с помощью регистровых операндов.

4.2 Динамическая символическая интерпретация бинарного кода архитектуры RISC-V 64

Разработанный метод динамической символической интерпретации опирается на метод конструирования символической семантики инструкций RISC-V 64, описанный в предыдущем подразделе. Для моделирования символического контекста архитектуры RISC-V 64 с помощью предлагаемого метода для бинарного кода задействованы следующие архитектурные компоненты:

- набор 64-битных регистров общего назначения $x0$ - $x31$, среди которых имеют служебное назначение регистр $x1$ (регистр возврата) и $x2$ (`sp`);
- указатель счетчика инструкций (`pc`);
- операционная семантика инструкций с целочисленными операндами;
- механизмы передачи управления, соглашения о вызовах и режимы адресации.

В отличие от `AArch64` инструкции архитектуры RISC-V обладают свойством лаконичности производимых операций. При этом сходство прослеживается в разграничении инструкций обращения к памяти и арифметических операций. В RISC-V не используются флаговые регистры, но, тем не менее, выполнение условных переходов возможно для набора соответствующих инструкций передачи управления после сравнения их операндов. Кроме того, для правильного моделирования потока управления и корректной обработки символической памяти стека необходимо детектировать псевдоинструкции для инструкций `JAL` и `JALR`.

Наиболее значимое отличие наблюдается для косвенных ветвлений в результате присутствия для архитектуры RISC-V технического ограничения в работе используемого инструментатора `DynamoRIO`. Для `AArch64` задача распознавания косвенных табличных переходов решается на уровне анализа базового блока, полностью загружаемого инструментатором перед обработкой входящих в него инструкций. Однако в ходе работы над

методом интерпретации бинарного кода RISC-V 64 оказалось, что фактически при появлении базового блока размером превышающем 5 инструкций, инструментатор разбивает его на несколько блоков. При этом из-за минималистичности, присущей компактному набору инструкций данной архитектуры, базовые блоки в среднем имеют более крупный размер по сравнению, например, с AArch64 и, тем более x86. Таким образом, базовые блоки, содержащие неявные табличные переходы могут быть разбиты на 2 или 3 части, каждая из которых содержит не больше 5 инструкций. Предикат же для инвертирования табличных переходов создается при обработке инструкции загрузки данных (load), но добавляется к общему предикату пути в процессе интерпретации инструкции передачи управления. Если две эти связанные инструкции попадают при разбиении в различные блоки, то адрес перехода заранее неизвестен. Поэтому вместо него записывается фиктивный адрес, а именно адрес целевой инструкции загрузки данных, увеличенный на единицу. Этот адрес является нечетным, так что из-за выравнивания он не сможет совпасть с каким-либо другим адресом инструкции, для которой не требуется добавление предиката, даже если это будет сокращенная инструкция. По нему инструкция перехода из другой части базового блока сможет определить нужный предикат.

Подробное объяснение механизма работы таблиц переходов с чтением из памяти как целевых адресов в исходном коде, так и относительных смещений приведено в аналогичном разделе для AArch64. Для архитектуры RISC-V 64 о присутствии в базовом блоке табличного перехода свидетельствуют следующие три составляющие:

- паттерн из трёх инструкций, которые могут располагаться в различном порядке: auipc – операция загрузки адресного значения, операция битового сдвига (например, slli), а также одна из инструкций сложения;
- инструкция чтения из памяти, которая располагается после всех трех инструкций, образующих паттерн;
- инструкция передачи управления для регистрового значения.

На рис. 4а-4в представлены варианты содержащих табличные переходы базовых блоков, которые будут подвергнуты разным разбиениям с точки зрения необходимых составляющих. На основе эвристических наблюдений можно сказать, что три образующие паттерн инструкции попадают в первый блок, в который также может попасть целевая инструкция чтения из памяти.

В следствие возможности разбиения базового блока на три части, каждая из которых содержит только одну из составляющих, характеризующих наличие косвенного ветвления, при появлении нового блока инструкций статус стадии поиска табличного перехода может быть различным. Для его хранения используется специально выделенная область памяти, которая проверяется и заполняется по мере появления составляющих.

В ситуации, когда производится первичное сканирование сначала проверяется, что последняя инструкция блока не относится к условным инструкциям передачи управления. Далее с конца осуществляется поиск последней инструкции загрузки данных (load) в блоке, при появлении которой будет использован статус хранилища относительно наличия паттерна в предыдущем блоке. Для первичного сканирования, при котором текущий инспектируемый блок действительно является началом базового блока, очевидно, статуса присутствия паттерна быть не может, поэтому требуется далее искать комбинацию из трех инструкций, что происходит и при отсутствии инструкции чтения в текущем блоке. При обнаружении данного паттерна сохраняется местоположение последней входящей в него инструкции для возможного сравнения с адресом найденной инструкции чтения. Это требуется для исключения случаев, когда в том же блоке разбиения присутствует нецелевая инструкция обращения к памяти, как на рис. 4а и рис. 4в. Если оба компонента успешно найдены, но последняя инструкция не передает управление по значению регистра (и, соответственно, вообще не является инструкцией передачи управления), то соответствующий статус вместе с

фиктивным адресом целевой инструкции чтения сохраняется в хранилище. При сканировании следующей части базового блока в результате обращения к хранилищу, которое происходит перед поиском инструкции чтения, будет возвращен адрес для добавления предиката. Если при первичном сканировании был найден только паттерн, то информация об этом будет также сохранена. Для разбиения на три части обновление статуса хранилища будет произведено дважды. При получении нужного фиктивного адреса целевой инструкцией передачи управления в символьном исполнителе будет заново создан предикат с правильным адресом.

| | | | |
|------|----------|-------|---------------------------------------|
| 6f6: | 47a5 | li | a5,9 |
| 6f8: | 0ae7e463 | bltu | a5,a4,7a0 <main+0xfc> |
| 6fc: | fec46783 | lwu | a5,-20(s0) |
| 700: | 00279713 | slli | a4,a5,0x2 |
| 704: | 00000797 | auipc | a5,0x0 |
| 708: | 19c78793 | addi | a5,a5,412 # 8a0 <_IO_stdin_used+0x78> |
| 70c: | 97ba | add | a5,a5,a4 |
| 70e: | 439c | lw | a5,0(a5) |
| 710: | 0007871b | sxt.w | a4,a5 |
| 714: | 00000797 | auipc | a5,0x0 |
| 718: | 18c78793 | addi | a5,a5,396 # 8a0 <_IO_stdin_used+0x78> |
| 71c: | 97ba | add | a5,a5,a4 |
| 71e: | 8782 | jr | a5 |
| 720: | 4785 | li | a5,1 |

Рис. 4а. Базовый блок с разбиением на 3 части.

| | | | |
|------|----------|-------|---|
| 718: | 12b00713 | li | a4,299 |
| 71c: | 00d77463 | bgeu | a4,a3,724 <main+0x80> |
| 720: | 33f0106f | j | 225e <main+0x1bba> |
| 724: | 00279713 | slli | a4,a5,0x2 |
| 728: | 00003797 | auipc | a5,0x3 |
| 72c: | ea078793 | addi | a5,a5,-352 # 35c8 <_IO_stdin_used+0x12e0> |
| 730: | 97ba | add | a5,a5,a4 |
| 732: | 439c | lw | a5,0(a5) |
| 734: | 0007871b | sxt.w | a4,a5 |
| 738: | 00003797 | auipc | a5,0x3 |
| 73c: | e9078793 | addi | a5,a5,-368 # 35c8 <_IO_stdin_used+0x12e0> |
| 740: | 97ba | add | a5,a5,a4 |
| 742: | 8782 | jr | a5 |
| 744: | 6042722 | sw | a5,a5,-20(s0) |

Рис. 4б. Базовый блок с разбиением, отделяющим инструкцию перехода.

| | | | |
|------|----------|-------|---------------------------------------|
| 800: | 4795 | li | a5,5 |
| 802: | 00e7da63 | bge | a5,a4,816 <main+0x6e> |
| 806: | 00000517 | auipc | a0,0x0 |
| 80a: | 10a50513 | addi | a0,a0,260 # 910 <_IO_stdin_used+0x80> |
| 80e: | e13ff0ef | jal | ra,620 <puts@plt> |
| 812: | 4781 | li | a5,0 |
| 814: | a821 | j | 82c <main+0x84> |
| 816: | 00001717 | auipc | a4,0x1 |
| 81a: | 7f270713 | addi | a4,a4,2034 # 2008 <jump_table> |
| 81e: | fec42783 | lw | a5,-20(s0) |
| 822: | 070e | slli | a5,a5,0x3 |
| 824: | 97ba | add | a5,a5,a4 |
| 826: | 639c | ld | a5,0(a5) |
| 828: | 9782 | jalr | a5 |

Рис. 4в. Базовый блок с разбиением, отделяющим блок с паттерном и нецелевой load-инструкцией от блока с целевой load-инструкцией и инструкцией передачи управления.

Определение размеров таблицы переходов для RISC-V 64 производится с помощью выбора максимального из сравниваемых аргументов инструкции условного выполнения, с помощью которой поток управления попал в блок, содержащий паттерн. Для выделенных возможных целевых адресов передачи управления также проверяется, что они относятся к исполняемому коду.

5. Экспериментальная оценка предлагаемых методов динамической символьной интерпретации

5.1 Байкал-М (ARM/AArch64)

Для апробации предложенного метода динамической символьной интерпретации были разработаны тестовые наборы на основе синтетических тестов и реальных приложений Байкал-М (AArch64). Метод был реализован на базе инструмента Sydr. Наиболее близким к нему по принципу работы и набору поддерживаемых символьных техник анализа среди открытых современных аналогов, поддерживающих архитектуру ARM/AArch64, можно назвать динамический интерпретатор SymQEMU. С целью оценки разработанного метода было проведено экспериментальное сравнение двух данных символьных анализаторов. Сравнение проводилось с помощью набора открытых приложений, запускаемых на процессоре Байкал-М. В качестве ограничения времени для каждого приложения был использован лимит в 20 минут. Результаты сравнения представлены в табл. 1.

Табл. 1. Сравнение динамических символьных интерпретаторов Sydr и SymQEMU для набора приложений архитектуры AArch64.

| Приложение | Sydr | | | | SymQEMU | | | |
|------------|----------|-------|----------|-------|----------|-------|----------|--------|
| | Покрытие | Уник. | Вх.файлы | Время | Покрытие | Уник. | Вх.файлы | Время |
| freetype2 | 98.93% | 227 | 1645 | 20m | 97.99% | 120 | 1287 | 20m |
| jsoncpp | 100% | 176 | 907 | 11,4s | 75.35% | 0 | 193 | 4,2s |
| lcms | 99.78% | 5 | 103 | 9,7s | 99.64% | 3 | 422 | 16m35s |
| libpng | 97.60% | 76 | 317 | 20m | 97.47% | 72 | 734 | 20m |
| libxml2 | 96.94% | 12 | 971 | 20m | 99.85% | 249 | 1736 | 20m |
| openthread | 99.96% | 12 | 285 | 1m26s | 99.89% | 5 | 486 | 2m51s |
| re2 | 93.37% | 219 | 47 | 5,6s | 89.64% | 140 | 81 | 36,1s |
| woff2 | 100% | 163 | 239 | 20m | 93.52% | 0 | 408 | 20m |

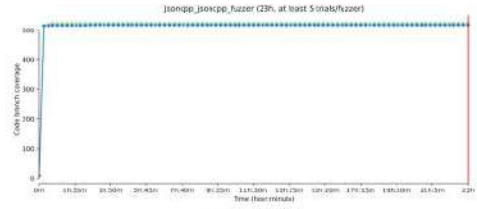
Результаты работы для каждого инструмента включают количество успешно сгенерированных тестов (обозначенных как "Вх. Файлы"), а также число уникальных строк покрытия, достигаемого для всего набора тестов (столбец "Уник."), которые были получены от обоих инструментов. Процент метрики покрытия считается для тестовых данных отдельного инструмента относительно общего достигнутого покрытия. Как можно заметить, для подавляющего большинства проектов (за исключением libxml2), предлагаемый метод символьной интерпретации продемонстрировал лучшие результаты.

Более того, метод успешно продемонстрировал свою эффективность для применения в контексте гибридного фаззинг-тестирования. Результаты в виде графиков метрики покрытия представлены на рис. 5. При проведении сравнения использовалась парная работа динамического символьного интерпретатора Sydr с инструментом фаззинга с обратной связью по покрытию (на графиках отмечены зеленым цветом). Пара состязалась с тем же фаззером, работающим в два потока (синий цвет). При осуществлении экспериментального сравнения для гибридного фаззинга применялась инфраструктура оценки фаззинг-инструментов FuzzBench. Ограничение времени, отведенного на фаззинг-тестирование, составляет 23 часа. По результатам для двух различных инструментов фаззинга libFuzzer и AFL++ (графики для которых на изображении относятся к левому и правому столбцам, соответственно), применение гибридного фаззинга вместо масштабирования приводило к увеличению достигнутой метрики покрытия более чем для половины исследуемых приложений.

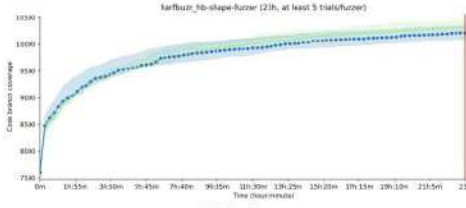
К факторам оценки работоспособности предложенного метода можно также отнести несколько выявленных и исправленных ошибок в интерфейсах символьного фреймворка Triton для работы с архитектурой ARM/AArch64.



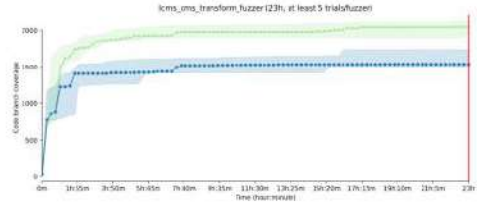
(a) freetype2



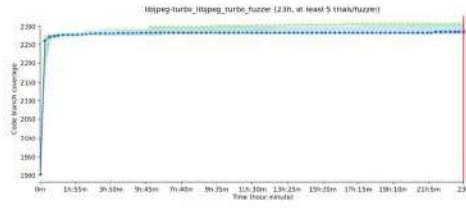
(a) jsonpp



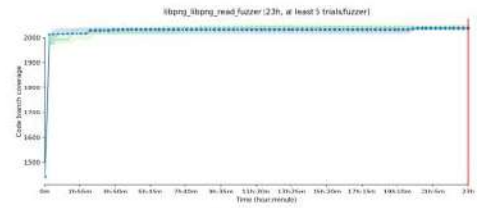
(b) harfbuzz



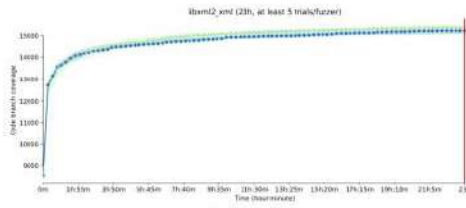
(b) lcms



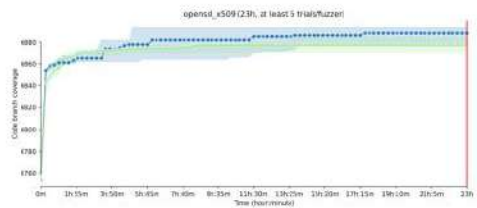
(c) libjpeg_turbo



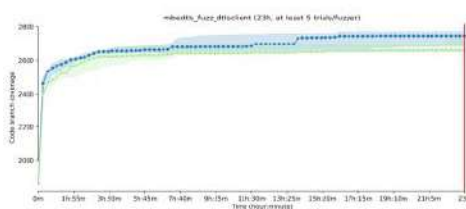
(c) libpng



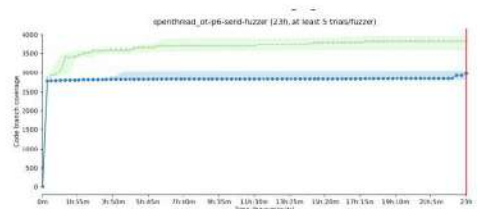
(d) libxml2



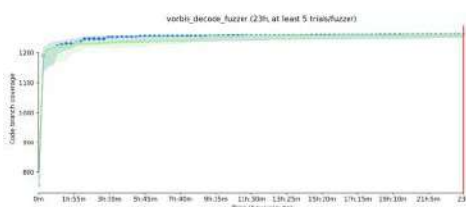
(d) openssl



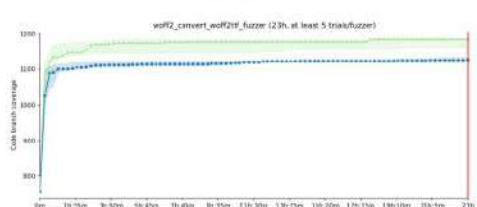
(e) mbedtls



(e) openthread



(f) vorbis



(f) woff2

Рис. 5. Графики достигнутых метрик покрытия FuzzBench.

5.2 RISC-V 64

Для оценки работоспособности предложенного метода динамической символьной интерпретации, внедренного в инструмент Sydr, были созданы синтетические тесты и тестовый набор на основе реальных приложений RISC-V 64. Для данного метода также было проведено экспериментальное сравнение Sydr с динамическим интерпретатором SymQEMU. Результаты сравнения представлены в табл. 2. Сравнение проводилось с помощью набора открытых приложений, запускаемых с помощью виртуальной машины, в связи с чем лимит времени для каждого приложения был увеличен до 60 минут. Метод проведения сравнения и условные обозначения для уникальных строк покрытия и процентного соотношения покрытия отдельного инструмента относительно общего достигнутого числа строк такие же как в предыдущем разделе. Из табл. 2 можно видеть, что для двух третей анализируемых приложений Sydr достигает лучших показателей достигнутого покрытия, а в остальных случаях показывает равные результаты для данной метрики.

Табл. 2. Сравнение динамических символьных интерпретаторов Sydr и SymQEMU для набора приложений архитектуры RISC-V 64.

| Приложение | Sydr | | | | SymQEMU | | | |
|------------|----------|-------|----------|--------|----------|-------|----------|--------|
| | Покрытие | Уник. | Вх.файлы | Время | Покрытие | Уник. | Вх.файлы | Время |
| ffmpeg | 100% | 0 | 28 | 2m11s | 100% | 0 | 197 | 5m11s |
| freetype2 | 98.99% | 445 | 2215 | 60m | 96.09% | 115 | 1042 | 60m |
| jsoncpp | 99.50% | 9 | 183 | 1m20s | 98.88% | 4 | 211 | 14s |
| lcms | 100% | 0 | 98 | 1m33s | 100% | 0 | 325 | 22m1s |
| openjpeg | 100% | 12 | 140 | 1m28s | 99.77% | 0 | 340 | 49m46s |
| openssl | 99.86% | 47 | 169 | 60m | 99.46% | 12 | 167 | 25m44s |
| re2 | 100% | 0 | 8 | 1m31s | 100% | 0 | 4 | 5s |
| woff2 | 99.82% | 111 | 475 | 57m27s | 95.93% | 5 | 645 | 60m |
| zlib | 100% | 698 | 160 | 45m16s | 66.25% | 0 | 77 | 41m58s |

Сравнительный запуск для гибридного фаззинга посредством инфраструктуры FuzzBench не проводился по причине использования виртуальной машины. Для данной архитектуры было проведено сравнение в режиме ручного тестирования между гибридным фаззингом пары Sydr и libFuzzer, а также двух параллельно работающих процессов libFuzzer. В Таблице 3 для каждого проекта представлены усредненные значения достигнутого покрытия по строкам среди 5 запусков, на каждый из которых отводилось по 24 часа. Как можно заметить, для 3 из 4 проектов гибридный фаззинг достиг более высоких значений метрики покрытия.

Табл. 3. Усредненные значения метрики покрытия по строкам для гибридного фаззинга и фаззинга «методом серого ящичка».

| Проект | Sydr + libFuzzer | 2 x libFuzzer |
|----------|------------------|---------------|
| lcms | 4527 | 4535,6 |
| openjpeg | 9257,4 | 9100,6 |
| woff2 | 3170 | 2991,6 |
| zlib | 2246,4 | 2155,2 |

6. Заключение

В данной работе были представлены разработанные методы динамической символьной интерпретации бинарного кода Байкал-М (AArch64) и RISC-V64, основанные на конструировании символьных выражений для операционной семантики машинных инструкций и применимые в контексте гибридного фаззинга. Данные методы были реализованы в инструменте Sydr, входящем в фреймворк динамического анализа Sydr-Fuzz [14]. Методы позволяют обнаруживать косвенные табличные переходы и определять соответствующие целевые адреса исполняемого кода. Помимо этого, был разработан метод

символической интерпретации набора целочисленных инструкций архитектуры RISC-V, включающего сокращенные инструкции и псевдоинструкции. Указанный метод был реализован в открытом фреймворке символической интерпретации Triton и может быть использован сообществом разработчиков для создания новых инструментов динамического анализа.

Список литературы / References

- [1]. ГОСТ Р 58412-2019: Защита информации. Разработка безопасного программного обеспечения. Угрозы безопасности информации при разработке программного обеспечения. — Национальный стандарт РФ, 2019.
- [2]. Serebryany, K. Continuous Fuzzing with libFuzzer and AddressSanitizer [Текст] / Kosta Serebryany // 2016 IEEE Cybersecurity Development (SecDev) / IEEE. 2016, с. 157.
- [3]. Fioraldi, A. AFL++: Combining Incremental Steps of Fuzzing Research [Текст] / A. Fioraldi, D. Maier, H. Eißfeldt, M. Heuse // 14th USENIX Workshop on Offensive Technologies (WOOT 20). 2020, с. 10.
- [4]. Molnar, D. Automated whitebox fuzz testing [Текст] / D. Molnar, P. Godefroid, M. Levin // Network and Distributed System Security Symposium, NDSS. 2008, с. 416-426.
- [5]. FuzzBench (Google). DSE+Fuzzing Experiment Report. 2021. [Электронный ресурс]. — URL: <https://www.fuzzbench.com/reports/experimental/2021-07-03-symbolic/index.html> (доступ 23.09.2025).
- [6]. Yun I. QSYM: A practical concolic execution engine tailored for hybrid fuzzing [Текст] / I. Yun [и др.] // 27th USENIX Security Symposium (USENIX Security 18). 2018, с. 745-761.
- [7]. Vishnyakov, A. Sydr: Cutting edge dynamic symbolic execution [Текст] / A. Vishnyakov [и др.] // 2020 Ivannikov ISPRAS Open Conference (ISPRAS). IEEE. 2020, с. 46-54.
- [8]. David, R. From source code to crash test-cases through software testing automation [Текст] / Robin David, Jonathan Salwan, Justin Bourroux // CESAR 2021: Automation in Cybersecurity. 2021.
- [9]. Cadar C. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. [Текст] / C. Cadar, D. Dunbar, D. R. Engler [и др.] // OSDI. T. 8. 2008, с. 209-224.
- [10]. Poeplau, S. SymQEMU: Compilation-based symbolic execution for binaries. [Текст] / S. Poeplau, A. Francillon // NDSS. 2021.
- [11]. Kutz D. Towards Symbolic Pointers Reasoning in Dynamic Symbolic Execution [Текст] / D. Kuts // 2021 Ivannikov Memorial Workshop (IVMEM). IEEE. 2021, с. 42-49.
- [12]. Vishnyakov A. Symbolic Security Predicates: Hunt Program Weaknesses [Текст] / A. Vishnyakov [и др.] // 2021 Ivannikov Ispras Open Conference (ISPRAS). IEEE. 2021, с. 76-85.
- [13]. Saudel, F. Triton: A Dynamic Symbolic Execution Framework [Текст] / Florent Saudel, Jonathan Salwan // Symposium sur la sécurité des technologies de l'information et des communications. SSTIC. 2015, с. 31-54.
- [14]. Vishnyakov, A. Sydr-Fuzz: Continuous Hybrid Fuzzing and Dynamic Analysis for Security Development Lifecycle [Текст] / A. Vishnyakov, D. Kuts, V. Logunova, D. Parygina, E. Kobrin, G. Savidov, A. Fedotov // 2022 Ivannikov ISPRAS Open Conference (ISPRAS). IEEE, 2022, с. 111-123.

Информация об авторах / Information about authors

Влада Игоревна ЛОГУНОВА – сотрудник отдела компиляторных технологий Института системного программирования. Сфера научных интересов: динамический анализ, анализ бинарного кода, динамическая символическая интерпретация, гибридный фаззинг.

Vlada Igorevna LOGUNOVA – Research Fellow at the Department of Compiler Technologies at the Institute for System Programming. Research interests: dynamic analysis, binary code analysis, dynamic symbolic execution, hybrid fuzzing.



Sydr-Fuzz: непрерывный гибридный фаззинг и динамический анализ для жизненного цикла безопасной разработки

А.В. Вишняков, ORCID: 0000-0003-1819-220X <pmvishnya@gmail.com>

¹*Д.О. Куц, ORCID: 0000-0002-0060-8062 <kutz@ispras.ru>*

^{1,2}*В.И. Логунова, ORCID: 0000-0002-3877-1906 <vlada@ispras.ru>*

^{1,3}*Д.А. Парыгина, ORCID: 0000-0002-4029-0853 <pa_darochek@ispras.ru>*

И.А. Кобрин, ORCID: 0000-0002-6035-0577 <ilayko3110@gmail.com>

Г.А. Савидов, ORCID: 0000-0000-0000-0000 <gsavidov@gmail.com>

А.Н. Федотов, ORCID: 0000-0002-8838-471X <splashgitar@gmail.com>

¹ *Институт системного программирования им. В.П. Иванникова РАН,
Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.*

² *Московский физико-технический институт,
Россия, 141701, Московская область, г. Долгопрудный, Институтский переулок, д. 9.*

³ *Московский государственный университет имени М.В. Ломоносова,
Россия, 119991, Москва, Ленинские горы, д. 1.*

Аннотация. Сегодня автоматизированные фреймворки динамического анализа для непрерывного тестирования востребованы как никогда – они обеспечивают безопасность программного обеспечения и соответствие требованиям Security Development Lifecycle (SDL). Эффективность поиска уязвимостей с помощью современных гибридных методов фаззинга превосходит традиционный фаззинг, основанный на покрытии кода. Мы предлагаем улучшенный конвейер динамического анализа для повышения результативности автоматизированного поиска ошибок, основанный на гибридном фаззинге. Его реализация – инструмент Sydr-Fuzz, в котором наш инструмент символьного выполнения Sydr интегрирован с libFuzzer и AFL++. В Sydr-Fuzz также входят проверка предикатов безопасности, инструмент triage для крашей Csr, а также утилиты минимизации корпуса и сбора покрытия. Бенчмаркинг показал, что Sydr-Fuzz превосходит фаззеры, ориентированные на покрытие, и сопоставим с современными гибридными фаззерами. В рамках проекта OSS-Sydr-Fuzz мы обнаружили 85 новых уязвимостей в реальных программах. Кроме того, мы открыли исходный код Csr для сообщества.

Ключевые слова: динамический анализ; гибридный фаззинг; непрерывный фаззинг; сортировка аварийных завершений; динамическая символьная интерпретация; ДСИ; поиск ошибок; цикл разработки безопасного кода; РБПО; информационная безопасность.

Для цитирования: Вишняков А.В., Куц Д.О., Логунова В.И., Парыгина Д.А., Кобрин И.А., Савидов Г.А., Федотов А.Н. Sydr-Fuzz: непрерывный гибридный фаззинг и динамический анализ для жизненного цикла безопасной разработки. Труды ИСП РАН, том 37, вып. 4, часть 2, 2025 г., стр. 251–270. DOI: 10.15514/ISPRAS–2025–37(4)–30.

Sydr-Fuzz: Continuous Hybrid Fuzzing and Dynamic Analysis for Security Development Lifecycle

A.V. Vishnyakov, ORCID: 0000-0003-1819-220X <pmvishnya@gmail.com>

¹ D.O. Kuts, ORCID: 0000-0002-0060-8062 <kutz@ispras.ru>

^{1,2} V.I. Logunova, ORCID: 0000-0002-3877-1906 <vlada@ispras.ru>

^{1,3} D.A. Parygina, ORCID: 0000-0002-4029-0853 <pa_darochek@ispras.ru>

E.A. Kobrin, ORCID: 0000-0002-6035-0577 <ilayko3110@gmail.com>

G.A. Savidov, ORCID: 0000-0000-0000-0000 <gsavidov@gmail.com>

A.N. Fedotov, ORCID: 0000-0002-8838-471X <splashgitar@gmail.com>

¹ *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*

² *Moscow Institute of Physics and Technology, Institutskiy Pereulok, 9, Dolgoprudny, Moscow Oblast, 141701, Russia.*

³ *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia.*

Abstract. Nowadays automated dynamic analysis frameworks for continuous testing are in high demand to ensure software safety and satisfy the security development lifecycle (SDL) requirements. The security bug hunting efficiency of cutting-edge hybrid fuzzing techniques outperforms widely utilized coverage-guided fuzzing. We propose an enhanced dynamic analysis pipeline to leverage productivity of automated bug detection based on hybrid fuzzing. We implement the proposed pipeline in the continuous fuzzing toolset Sydr-Fuzz which is powered by hybrid fuzzing orchestrator, integrating our DSE tool Sydr with libFuzzer and AFL++. Sydr-Fuzz also incorporates security predicate checkers, crash triaging tool Casr, and utilities for corpus minimization and coverage gathering. The benchmarking of our hybrid fuzzer against alternative state-of-the-art solutions demonstrates its superiority over coverage-guided fuzzers while remaining on the same level with advanced hybrid fuzzers. Furthermore, we approve the relevance of our approach by discovering 85 new real-world software flaws within the OSS-Sydr-Fuzz project. Finally, we open Casr source code to the community to facilitate examination of the existing crashes.

Keywords: dynamic analysis; hybrid fuzzing; continuous fuzzing; crash triage; dynamic symbolic execution; DSE; error detection; security development lifecycle; SDL; computer security.

For citation: Vishnyakov A.V., Kuts D.O., Logunova V.I., Parygina D.A., Kobrin E.A., Savidov G.A., Fedotov A.N. Sydr-Fuzz: Continuous Hybrid Fuzzing and Dynamic Analysis for Security Development Lifecycle. *Trudy ISP RAN/Proc. ISP RAS*, vol. 37, issue 4, part 2, 2025, pp. 251-270 (in Russian). DOI: 10.15514/ISPRAS-2025-37(4)-30.

1. Введение

Современное промышленное программное обеспечение (ПО) становится всё сложнее и охватывает практически все сферы жизни – от незначительных до критически важных. Введение процесса разработки безопасного ПО (РБПО) [1-3] стало естественной практикой для большинства компаний-разработчиков. Одним из широко применяемых методов анализа является гибридный фаззинг [4-12], сочетающий фаззинг и динамическое символьное выполнение (ДСИ). Фаззинг быстро расширяет покрытие кода, преодолевая простые ограничения, но плохо исследует сложные участки. ДСИ, наоборот, хорошо справляется с нетривиальными ветвлениями, но работает медленнее. Совместный запуск фаззинга и ДСИ позволяет достичь лучших результатов, чем фаззинг, основанный только на покрытии [13].

Мы разработали новый инструмент гибридного фаззинга, основанный на нашем символьном исполнителе Sydr [14], интегрировав его с AFL++ [15] и libFuzzer [16]. Такой подход позволяет повысить эффективность анализа по сравнению с существующими решениями.

В современном быстроразвивающемся мире важно не только находить ошибки, но и своевременно их исправлять. Интеграция различных инструментов в единый комплекс позволяет значительно повысить продуктивность. На этой основе мы создали набор инструментов Sydr-Fuzz, объединяющий гибридный фаззинг, минимизацию корпуса, проверку предикатов безопасности, сортировку аварийных завершений и сбор покрытия. Sydr-fuzz реализует удобный и продуктивный конвейер динамического анализа. Последовательный запуск этапов конвейера sydr-fuzz позволяет максимально эффективно использовать возможности динамического анализа на основе гибридного фаззинга. Гибридный фаззинг выполняется на начальном этапе конвейера. Его результатом становится корпус тестовых входных данных, которые обеспечивают новое покрытие кода и потенциально могут привести к обнаружению ранее неизвестных ошибок. Так как корпус может содержать множество входных данных, открывающих одно и то же покрытие или ошибки, полезно удалить избыточные входные данные из дальнейшего анализа. Минимизация корпуса стремится удалить как можно больше тестов, сохранив при этом то же покрытие кода, чтобы оставить лишь наиболее ценные входные данные.

После первых двух шагов мы получаем корпус адекватного размера, содержащий набор тестов, который можно затем обработать разными способами. Sydr-fuzz предлагает три стратегии для извлечения информации из результатов гибридного фаззинга. Первая – обнаружение ошибок с помощью символьных предикатов безопасности. Эта техника основана на динамическом символьном исполнении с дополнительными ограничениями, нацеленными на выявление четырёх видов ошибок: разыменованное нулевого указателя, деление на ноль, переполнение целого числа и выход за границы массива. Некоторые из этих видов ошибок связаны с несколькими категориями CWE. Вторая стратегия – сбор покрытия кода, обеспечиваемого корпусом. И третья включает сортировку аварийных завершений с использованием инструментов Casr [17]. Эти инструменты позволяют генерировать отчёты о сбоях, вызванных входными данными из корпуса, затем выполнять дедупликацию и кластеризацию этих отчётов. На выходе результаты представляются в виде кластеров, соответствующих потенциально разным багам, дополненным соответствующими тестовыми случаями.

Данная работа содержит следующие основные результаты:

- Мы разработали инфраструктуру непрерывного гибридного фаззинга для эффективного динамического анализа программ. Мы объединили этапы гибридного фаззинга [4–12], минимизации корпуса, сбора покрытия и классификации сбоев [17], применения символьных предикатов безопасности [18] в инструменте sydr-fuzz и предлагаем конвейер динамического анализа для максимизации его эффективности. Мы представляем репозиторий OSS-Sydr-Fuzz [19], созданный по образцу OSS-Fuzz [20] и адаптированный для гибридного фаззинга с использованием sydr-fuzz.
- Мы разработали новый гибридный фаззер на основе инструмента ДСИ Sydr [14]. Мы интегрировали Sydr с AFL++ [15] и libFuzzer [16] (что является первой интеграцией между инструментом ДСИ и libFuzzer). Мы объединили возможности нашего динамического символьного исполнителя Sydr с современными фаззерами. Мы реализовали в Sydr ряд полезных функций, которые способствуют достижению лучших результатов оценки. Мы внедрили анализ символьных указателей в процессе гибридного фаззинга [21].
- Мы провели оценку sydr-fuzz на платформе Google FuzzBench [22] в сравнении с современными фаззерами с покрытием и гибридными фаззерами. Мы демонстрируем, что sydr-fuzz превосходит фаззеры с покрытием и оказывается сопоставим с гибридными фаззерами, получая значительную пользу от своего символьного исполнителя Sydr [23].

- Мы открыли исходный код инструмента Casr [17] для кластеризации отчетов о сбоях, дедупликации и оценки серьезности [24].

В основном данная статья является переводом доклада Vishnyakov, A. Sydr-Fuzz: Continuous Hybrid Fuzzing and Dynamic Analysis for Security Development Lifecycle [Текст] / A. Vishnyakov, D. Kuts, V. Logunova, D. Parygina, E. Kobrin, G. Savidov, A. Fedotov // 2022 Ivannikov ISPRAS Open Conference (ISPRAS). IEEE, 2022. — С. 111–123. При переводе были устранены некоторые неточности.

2. Обзор существующих решений

2.1 Гибридный фаззинг

Гибридный фаззинг является современной методикой обнаружения программных ошибок. Его сила заключается в легковесном и быстром фаззинге и точном символьном выполнении. Фаззинг помогает быстро обнаруживать новые пути, в то время как ДСИ отвечает за систематическое исследование кода.

2.1.1 QSYM

Гибридный фаззер QSYM [7] стал одним из первых инструментов, показавших эффективность гибридного фаззинга. Yun и др. реализовали гибридный фаззер, достаточно легковесный, чтобы позволить ДСИ и фаззеру работать параллельно (в то время как в Driller [6] ДСИ запускается на короткий период для помощи фаззеру, когда тот перестает открывать новое покрытие). QSYM использует динамическую бинарную трансляцию для сокращения количества символьных эмулируемых инструкций и отказывается от использования промежуточного представления, чтобы исключить дополнительные накладные расходы. Авторы предложили две техники оптимизации для повышения эффективности анализа. Во-первых, отсечение базовых блоков позволяет пропускать эмуляцию некоторых блоков, если они выполнялись слишком часто. Во-вторых, оптимистичное решение предполагает решение только целевого ограничения, если весь предикат пути невыполним. Эти техники не обеспечивают полный анализ, но помогают инвертировать больше символических ветвей за единицу времени. Другая полезная техника, предложенная QSYM, – это кэш для инвертированных ветвей с двумя режимами кэширования. Статический режим означает, что каждая символическая ветвь инвертируется только один раз, в то время как контекстный режим позволяет учитывать глубину и набор ранее выполненных символических ветвей. QSYM также предлагает обработку символических адресов [4] путем их простого фаззинга. Он ищет минимальные и максимальные значения адресов с помощью SMT-решателя и генерирует новые входные данные при каждом вызове. Что касается гибридного фаззинга, QSYM запускает фаззер параллельно с символьным исполнителем и позволяет им обмениваться новыми тестовыми входными данными. ДСИ предпочитает такие входные файлы, которые недавно обнаружили новое покрытие и одновременно имеют меньший размер.

2.1.2 SymCC, SymQEMU

Авторы SymCC [8] разработали новую технику символьного выполнения на основе компиляции. Инструментирующий код для символьного выполнения вставляется в код целевого приложения, поэтому скомпилированный бинарный файл может быть выполнен без переключения между выполнением программного кода и интерпретатором. Код для обновления символьного состояния и обработки символьных вычислений генерируется только один раз во время компиляции. Этот метод также выигрывает от возможности применять все оптимизации LLVM IR и CPU. SymCC включает символьный бэкэнд в

библиотеки, используемые целевой программой. Проверки конкретности позволяют значительно сократить количество символьных обрабатываемых вычислений. Как продолжение SymCC, авторы представили SymQEMU [9], который предложил применять символьное выполнение на основе компиляции к бинарным файлам при отсутствии исходного кода. Такой метод сочетает высокую скорость анализа с архитектурной независимостью. SymQEMU был построен на основе QEMU путем расширения его компонента TCG так, что код символьной обработки вставляется в IR TCG ops перед компиляцией в машинный код хоста. Символьный анализ останавливается на границе системного вызова, что позволяет достичь лучшей производительности. SymQEMU также реализует сборщик мусора для символьных выражений для эффективного управления памятью. Как SymCC, так и SymQEMU поддерживают интеграцию с AFL++ [15] способом, аналогичным QSYM.

2.1.3 FUZZOLIC

Borzacchiello и др. [10] предложили подход, схожий с SymQEMU. Они значительно повысили эффективность символьного выполнения благодаря быстрому анализу, построенному на основе QEMU, и новому решателю Fuzzy-Sat [25], который использует фаззинг для решения запросов. FUZZOLIC выполняет JIT-компиляцию для добавления инструментирующего кода во время выполнения. Он компилирует каждый базовый блок только один раз и выигрывает от вставки инструментирующего кода в код целевой программы. FUZZOLIC состоит из трассировщика и решателя, которые работают в отдельных параллельных процессах, и это первое важное отличие от SymQEMU. Трассировщик выполняет программу и генерирует символьные выражения, которые отправляются решателю, ответственному за решение запросов. Процессы общаются через разделяемую память. Также, в отличие от SymQEMU, FUZZOLIC внедряет символьную инструментацию в целевой код после того, как QEMU сгенерировал TCG базового блока. Это позволяет воспользоваться преимуществами оптимизаций TCG на уровне всего базового блока. Последнее существенное отличие заключается в том, что SymQEMU обрабатывает только архитектурно-независимые TCG-инструкции, в то время как FUZZOLIC может вставлять символьную инструментацию для большого количества архитектурно-зависимых TCG-нативных хелперов для платформ x86 и x86-64. Схема гибридного фаззинга в FUZZOLIC аналогична QSYM. Он принимает входные данные, мутированные AFL++, в то время как фаззер принимает интересные входные данные, предоставляемые символьным исполнителем.

2.1.4 PASTIS

David и др. [12] представили инфраструктуру автоматизированного тестирования, сочетающую фаззинг и ДСИ для проверки предупреждений, полученных от некоторого инструмента статического анализа, и выявления ошибок, если это возможно. Информация, предоставляемая SAST-инструментом, используется для добавления встроенных функций в целевой код. Все варианты кода компилируются и отправляются в брокер PASTIS, который выполняет всю коммуникацию между механизмами тестирования. С одной стороны, Honggfuzz [26] представляет набор инструментов для фаззинга, а с другой стороны, фреймворк TritonDSE [27] отвечает за символьное выполнение.

2.1.4 SymSan, Jigsaw

Основная идея SymSan [11] заключается в построении конколического исполнителя как особой формы анализа динамических потоков данных. SymSan выполняет инструментирование кода на этапе компиляции в LLVM IR [28]. Использование

высокооптимизированной инфраструктуры из DFSAN [29] помогает снизить накладные расходы на хранение и извлечение символьных выражений, форма которых также оптимизирована. SymSan предлагает использовать таблицу AST вместе со специальным дизайном узлов AST для хранения символьных выражений. В сочетании с простым прямым выделением новых узлов это позволяет значительно снизить потери производительности при обработке символьных выражений. Дополнительно SymSan реализует дедупликацию хранимых узлов AST и упрощение операций загрузки и сохранения. Авторы реализовали гибридный фаззер на основе SymSan и Angora [30]. Они также предложили новую конструкцию для повышения пропускной способности поиска и внедрили ее в прототип Jigsaw [31], который используется в качестве решателя для гибридного фаззера. Суть подхода заключается в оценке вновь сгенерированных входных данных с помощью JIT-скомпилированных ограничений пути. Jigsaw компилирует предобработанные подзадачи AST в функции LLVM IR, использует JIT-движок LLVM для компиляции IR в нативную функцию и ищет удовлетворяющее решение с помощью градиентно-управляемого поиска.

2.2 Непрерывный фаззинг

Сближение безопасной разработки и фаззинга становится отраслевым стандартом [2, 3], подобно обязательному модульному тестированию. Непрерывный фаззинг – это подход к организации автоматизированного фаззинг-тестирования как рутинной процедуры, например, путем его интеграции в конвейер CI/CD. Инфраструктура непрерывного фаззинга может быть частью внутреннего рабочего процесса организации [32, 33] или существовать в форме инструмента «фаззинг-как-услуга» [19, 34–37]. Существующие решения включают как простые запускатели фаззинг-задач, так и крупные фреймворки, охватывающие расширенную функциональность для масштабируемого и ансамблевого [12, 38, 39] фаззинга, анализа и отчетности по обнаруженным уязвимостям, бисекции коммитов, вызвавших регрессию, и т.д.

- 1) *OSS-Fuzz & ClusterFuzz*: Хотя автоматизированное фаззинг-тестирование не является серебряной пулей, оно достаточно эффективно для обнаружения ошибок и сокращения требуемых человеческих усилий аналитиков. Например, ClusterFuzz [40] уже нашел более 25 000 ошибок в продуктах, разработанных Google, и более 43 500 ошибок [41] в программном обеспечении с открытым исходным кодом, включенном в OSS-Fuzz. Проект OSS-Fuzz [34] выборочно предоставляет сообществу открытого исходного кода доступ к масштабируемым мощностям инфраструктуры ClusterFuzz, работающей на Google Cloud Platform [42]. ClusterFuzz предназначен для автоматического выполнения любой задачи в рамках жизненного цикла фаззинга, за исключением написания фаззинг-целей и исправления ошибок. Это включает планирование и запуск заданий фаззинга, сбор статистики, дедупликацию и классификацию новых сбоев, минимизацию тестовых случаев, бисекцию коммита, вызвавшего регрессию, и проверку исправления ошибки. ClusterFuzz поддерживает несколько механизмов фаззинга с покрытием [15, 16, 26], черный ящик фаззинга и ряд санитайзеров. Поскольку ClusterFuzz не обладает собственной инфраструктурой сборки, типичный набор проекта для участия в OSS-Fuzz включает образ Docker, конфигурацию сборки и как минимум одну фаззинг-цель. На данный момент OSS-Fuzz непрерывно проводит фаззинг более 650 критически важных и широко используемых проектов с открытым исходным кодом.
- 2) *OneFuzz*: Другой фреймворк непрерывного фаззинга, OneFuzz [32], был открыт Microsoft. Подобно ClusterFuzz, OneFuzz в настоящее время применяется к программному обеспечению Microsoft (Windows, Edge и др.) и привязан к

- корпоративной облачной среде Azure [43]. Поддерживаемые механизмы фаззинга – это libFuzzer [16], AFL++ [15] и Radamsa [44]. OneFuzz предоставляет возможность воспользоваться встроенными шаблонами, а также позволяет создавать настраиваемые конвейеры фаззинга. Обнаруженные уязвимости классифицируются по стабильности воспроизведения и дедулицируются. Кроме того, OneFuzz позволяет проводить live-отладку сбоев и мониторинг рабочего процесса фаззинга с помощью настраиваемых веб-хуков.
- 3) *Grizzly*: FirefoxCI TaskCluster использует Grizzly [33] – масштабируемый фреймворк для фаззинга, специфичный для браузеров. Grizzly независимо запускает браузер и механизм фаззинга, управляет передачей данных между ними во время анализа и выполняет сокращение тестовых случаев. Два основных интерфейса, Target и Adapter, отвечают за развертывание желаемой комбинации браузера и фаззера, но фреймворк в первую очередь ориентирован на методы фаззинга черного ящика. Основной размер шрифта – 10 пт.;
 - 4) *Fuzzit*: Интегрированный в сервис Gitlab [45] Fuzzit [35] был разработан для интеграции фаззинга в систему непрерывной сборки проекта. Он представляет собой раннеры для набора механизмов фаззинга с покрытием, позволяющие проводить фаззинг для разных языков программирования (например, в случае C/C++ возможны варианты libFuzzer и AFL++), и способен проводить регрессионное тестирование.
 - 5) *syzbot*: Система непрерывного фаззинга ядра Linux syzbot [46] последовательно генерирует структурированные отчеты об обнаруженных сбоях ядра. Помимо поиска ошибок, система отслеживает устаревание отчетов об ошибках и проверяет исправленные проблемы. После проведения тестирования патча syzbot проверяет, что соответствующий коммит попал в сборки ядра для всех отслеживаемых ветвей, чтобы закрыть отчет об ошибке. Схожие по проявлениям сбои могут быть сообщены отдельно, вместо того чтобы объединяться с существующей проблемой.

3. Гибридный фаззинг

Статистика показывает, что гибридные фаззинговые механизмы, сочетающие мощь символического выполнения и фаззинга, могут достигать большего покрытия, чем два одновременно работающих фаззера [13]. Основываясь на этой идее, мы реализовали наш гибридный фаззинговый инструмент, объединив Sydr [14] с двумя наиболее популярными и мощными фаззерами с открытым исходным кодом: libFuzzer [16] и AFL++ [15]. Для достижения цели создания эффективного гибридного фаззера необходимо решить две основные задачи. Во-первых, мы разрабатываем различные эвристики, чтобы заставить Sydr работать максимально эффективно в сочетании с фаззером. Во-вторых, мы реализуем интеграцию Sydr и фаззеров, такую как межпроцессное взаимодействие и планирование входных данных для Sydr.

Для запуска sydr-fuzz в режиме гибридного фаззинга необходимо собрать целевой бинарный файл в двух версиях. Первая должна быть собрана с санитайзерами и фаззинговой инструментацией для использования соответствующим фаззером. Второй бинарный файл должен быть собран без какой-либо инструментации – он будет использоваться Sydr.

Мы реализовали различные эвристики и функции, чтобы Sydr находил новые интересные входные данные быстрее и работал эффективнее в сочетании с фаззерами. Например, мы реализовали кэш в Sydr, работающий аналогично кэшу QSYM, чтобы экономить время на попытках инвертировать одни и те же символические ветви. Основная идея заключается в том, что для каждой ветви вычисляется уникальный хэш, который используется как индекс

в битовой карте. Каждая отдельная ветвь соответствует байту, который является счетчиком, показывающим, сколько раз предпринималась попытка инвертировать ветвь. Более того, мы сохраняем контекст выполнения для каждой ветви, который отражает, какие ветви встречались во время выполнения до рассматриваемой. Если контекст изменяется (т.е. Sydr достигает ветви по другому пути выполнения), мы пытаемся инвертировать ветвь. В противном случае мы пытаемся инвертировать отдельную ветвь, когда соответствующий счетчик равен 0, 2, 4, 8 и так далее (степени двойки), пока он не достигнет 255, после чего мы прекращаем инверсию данной конкретной ветви.

Слайсинг [14] – это еще одна эвристика для ускорения символьного выполнения. Каждый раз, когда мы пытаемся инвертировать ветвь, мы применяем алгоритм слайсинга к ее предикату пути. Идея алгоритма заключается в том, что мы оставляем в предикате пути только те ограничения пути, которые зависят по данным от целевой ветви. Остальные ограничения пути устраняются, а входные данные для них берутся из исходного набора входных данных программы.

Мы реализовали оптимистичное и строгое оптимистичное решение [47], чтобы избежать неполных и излишне перегруженных формул. Таким образом, Sydr инвертирует больше ветвей и генерирует больше входных данных. Основная идея следующая. Когда исходный урезанный (sliced) предикат невыполним, мы строим оптимистичный предикат, состоящий только из ограничения целевой ветви. Если он выполним, мы строим строго оптимистичный предикат, который получается из исходного урезанного предиката путем исключения некоторых нерелевантных ограничений пути на основе анализа стека вызовов программы и управляющих зависимостей ветвей. Если строго оптимистичный предикат совпадает с оптимистичным или невыполним, мы сохраняем входной файл для оптимистичного решения. Если они не совпадают и строго оптимистичный предикат выполним, мы сохраняем оба сгенерированных набора входных данных.

Мы реализовали семантику функций для ускорения символьного выполнения функций стандартной библиотеки. Вместо того чтобы входить в тела функций стандартной библиотеки (таких как `strto*`, `*alloc`, `strcmp` и т.д.) и выполнять их символьно, мы строим символьные формулы для их возвращаемых значений, что помогает ускорить символьное выполнение и избежать переполнения ограничений.

Мы ограничиваем время, которое отводится процессу Sydr на решение одного запроса, и общее время, которое он может потратить на решение всех запросов, чтобы избежать ситуаций, когда SMT-решатель застревает на сложных запросах. Более того, мы устанавливаем таймаут на общее время выполнения одного процесса Sydr, чтобы избежать зависания программы.

Также стоит упомянуть, что Sydr имеет функциональность для инверсии таблиц переходов (операторов `switch`) и обработки символьных адресов. Существуют два способа обработки символических адресов в Sydr: полная обработка символических указателей и фаззинг символических адресов. Полная поддержка символических указателей [21] резко перегружает символьный движок, что приводит к ухудшению результатов фаззинга. Однако этот режим все же позволяет символьному движку успешно обнаруживать новое уникальное покрытие. Следовательно, оптимальная стратегия – включать этот режим периодически, а не использовать при каждом запуске Sydr. Когда Sydr работает без обработки символических указателей, он выполняет SMT-фаззинг для всех символических адресов. Это легковесный, но менее точный метод, который перебирает возможные значения символьного адреса с помощью SMT-решателя.

Перед началом процесса гибридного фаззинга исходный корпус автоматически минимизируется, чтобы оставить только те входные данные, которые приносят какое-либо новое покрытие.

4. Предикаты безопасности

Мы предлагаем метод предикатов безопасности – метод точного обнаружения ошибок на основе динамического символьного выполнения – как часть sydr-fuzz. Мы реализуем автоматическую проверку предикатов безопасности с последующей верификацией и дедупликацией входных данных, сгенерированных предикатами безопасности, для выявления ошибок.

Предикат безопасности – это булевый предикат, который принимает значение истина, если инструкция (или функция) программы вызывает ошибку [18]. Реализация предикатов безопасности является частью Sydr [14] и основана на следующей идее. Мы выполняем программу символьно с использованием набора входных данных, который не приводит к ошибке. Каждый раз, когда выполняется инструкция, оперирующая символьными данными, мы строим соответствующий предикат безопасности для проверки на определенный тип ошибки. Затем мы объединяем предикат безопасности с "нарезанными" ограничениями ветвей из предиката пути и передаем результирующий предикат SMT-решателю, такому как Bitwuzla [48], чтобы сгенерировать входные данные, которые воспроизведут ошибку. Если предикат выполним, мы сохраняем вход и сообщаем об ошибке. Предикаты безопасности могут обнаруживать несколько типов слабостей: разыменованное нулевого указателя, деление на ноль, целочисленное переполнение и выход за границы буфера.

Предикаты безопасности для разыменования нулевого указателя и деления на ноль работают схожим образом. В первом случае мы строим предикат безопасности, чтобы проверить, может ли символьный адрес быть равен нулю, каждый раз при выполнении инструкции доступа к памяти. Во втором случае мы строим предикат безопасности, чтобы проверить, может ли символьный делитель быть равен нулю, каждый раз при выполнении инструкции деления, такой как `div` или `idiv`, во время символьного выполнения.

Для ошибки выхода за границы мы строим предикат безопасности при каждой инструкции доступа к памяти. Этот предикат истинен, если символьный адрес может быть меньше нижней границы массива или больше верхней границы. Для определения границ массива мы ведем теневой стек и теневую кучу во время символьного выполнения. Мы сохраняем информацию о границах массивов, выделенных в куче, при каждом вызове функции `*alloc` и удаляем ее при вызове `free`. Мы изменяем теневой стек в соответствии со стеком вызовов, поскольку рассматриваем верхнюю границу массива в стеке как адрес начала фрейма функции. Для массивов в куче мы получаем их границы из теневой кучи. Для массивов в стеке мы считаем верхней границей точку вызова текущей функции и эвристически вычисляем нижнюю границу.

При каждой арифметической инструкции во время символьного выполнения мы строим предикат безопасности для ошибки целочисленного переполнения. Этот предикат истинен, если процессорные флаги переноса CF (Carry Flag) или переполнения OF (Overflow Flag) равны 1 после выполнения инструкции. Если результат знаковый, мы проверяем только флаг OF, иначе – CF. Знаковость определяется по ранее встреченным условным инструкциям, которые используют хотя бы один операнд, совпадающий с используемым анализируемой инструкцией. Например, `JL` указывает, что результат знаковый, и мы должны проверять только флаг OF. Ключевое отличие этого предиката от других в том, что мы разделяем понятия источника ошибки (арифметическая инструкция, где может возникнуть ошибка) и приемника ошибки (место, где ошибка может быть использована), и проверяем предикат безопасности для источника только когда найден приемник, использующий этот источник. Мы выделяем следующие типы приемников: инструкции ветвления, инструкции разыменования адреса и аргументы функций [49].

Мы реализуем автоматическую проверку предикатов безопасности в sydr-fuzz. Предикаты безопасности ищут ошибки только на пути, заданном исходным набором входных данных.

Следовательно, нам нужно достичь максимального покрытия минимальным количеством входных данных. Сначала мы запускаем гибридный фаззинг для достижения высокого покрытия. Затем мы выполняем минимизацию корпуса, чтобы оставить минимальное количество файлов, обеспечивающих то же покрытие, что и до минимизации. После минимизации запускается проверка предикатов безопасности. Результаты предикатов безопасности часто оказываются ложными срабатываниями, поэтому мы реализуем их автоматическую верификацию. Мы запускаем целевой бинарный исполняемый файл, собранный с санитайзерами (автоматические проверки на наличие ошибок, встраиваемые компилятором), на входном файле, сгенерированном предикатами безопасности. Если какой-либо санитайзер сообщает об ошибке в месте, указанном предикатами как источник, мы считаем входной файл верифицированным. В противном случае мы запускаем исполняемый файл с санитайзерами на исходном входном файле из корпуса, которое использовалось для проверки предикатов, и, если входной файл от предикатов вызывает новые предупреждения санитайзеров, мы считаем его верифицированным. Sydr-fuzz выполняет дедупликацию верифицированных результатов предикатов безопасности на основе исходного файла, строки и столбца кода, где обнаружена ошибка, чтобы упростить анализ результатов.

5. Сортировка аварийных завершений

Классификация аварийных завершений является важным этапом динамического анализа, включая фаззинг или гибридный фаззинг. Количество сбоев, которые генерирует фаззер, может быть значительным. На то, чтобы определить, какие сбои представляют одну и ту же ошибку, можно потратить много времени. Мы предлагаем подход, который должен помочь разработчикам тратить меньше времени на анализ и исправление различных ошибок и проблем. Наши инструменты Casr [17] позволяют создавать отчеты о сбоях, дедуплицировать и кластеризовать их. Основные этапы работы sydr-fuzz casr следующие:

- 1) casr-san запускает инструментированный бинарный файл на всех входных файлах, которые потенциально вызывают сбои, и генерирует отчеты о сбоях на основе отчетов санитайзеров (при необходимости с помощью gdb).
- 2) casr-cluster запускает алгоритм дедупликации для отчетов Casr, полученных от casr-san. Дедупликация основана на трассе стека: каждый фрейм хэшируется, затем хэш всей трассы стека добавляется в хэш-набор. В результате в хэш-наборе останутся только уникальные отчеты (подробности [17]), остальные будут удалены из каталога casr.
- 3) casr-cluster запускает иерархическую кластеризацию отчетов Casr. Расстояние между сбоями вычисляется на основе трассы стека [17].
- 4) casr-gdb генерирует отчеты о сбоях для неинструментированных целевых бинарных файлов на основе кластеров, полученных на третьем шаге.

В результате мы получаем кластеры, содержащие потенциально различные ошибки (в виде отчетов Casr). Вместе с каждым отчетом предоставляется соответствующий входной файл, которое приводит к этому сбою. Некоторые шаги (3, 4) можно пропустить, установив соответствующие опции.

Отчет о сбое содержит информацию о сбое, такую как версии ОС и пакетов, выполненная командная строка, трасса стека, открытые файлы и сетевые подключения, состояние регистров, часть исходного кода, вызвавшая сбой, с соответствующей строкой сбоя и т.д. Таким образом, разработчику не нужно запускать gdb и анализировать сбой вручную, вся необходимая информация уже есть в отчете.

Также наши инструменты Casr позволяют оценить серьезность сбоя. Мы делим их на три широких класса (как и в gdb exploitable [50]): эксплуатируемый (exploitable), вероятно эксплуатируемый (probably exploitable) и не эксплуатируемый (not exploitable). Классы 260

включают различные ошибки, которые могут возникнуть во время выполнения программы, такие как переполнение стека, двойное освобождение памяти и другие. Класс сбоя определяется на основе трассы стека, дизассемблированного участка кода, вызвавшего исключение, сигнала, поступившего в программу, и некоторой другой информации. Классифицированные сбои помогают разработчикам понять, какие из них следует анализировать и исправлять в первую очередь.

6. Инфраструктура непрерывного гибридного фаззинга

Интеграцию непрерывного фаззинга с использованием фреймворка sydr-fuzz можно продемонстрировать на примере проекта OSS-Sydr-Fuzz [19]. OSS-Sydr-Fuzz создан по образцу упомянутого выше OSS-Fuzz [34] и призван подтвердить эффективность гибридного фаззинга на реальном программном обеспечении. Соответствующий список обнаруженных уязвимостей [51] представлен в разделе 7.

Необходимые исходные коды включают код тестируемого проекта и дополнительный репозиторий фаззинга OSS-Sydr-Fuzz [19]. Репозиторий содержит набор предварительно подготовленных средств для настройки и запуска сессии гибридного фаззинга. Ключевыми задачами для запуска являются сборка кода проекта, развертывание инструментов фаззинга и подготовка фаззинг-целей. Соответственно, стандартный пакет проекта OSS-Sydr-Fuzz включает скрипт сборки, инструкции Dockerfile, составленные фаззинг-цели и файл конфигурации гибридного фаззинга для каждой цели. Кроме того, репозиторий может предоставлять дополнительные материалы, такие как начальный корпус входных данных, словари и т.д.

Изображенный на схеме (рис. 1) процесс CI фаззинга реализован на платформе GitLab [52]. Система применяет конвейер динамического анализа, состоящий из пяти этапов: гибридный фаззинг, минимизация корпуса, поиск ошибок с помощью предикатов безопасности, сортировка аварийных завершений и сбор покрытия. Фаззинг запускается при возникновении внешнего триггерного события. В настоящее время он запускается вручную для выбранного проекта и ветки. Альтернативно, он может быть организован как запланированная рутинная операция или запускаться новыми коммитами.

Перед началом анализа выбранного проекта запускатель заданий фаззинга активирует задачу сборки Docker. В первую очередь, сборщик проверяет актуальность коммита и запрашивает у реестра контейнеров, существует ли образ Docker из предыдущей итерации и можно ли его повторно использовать. Полученный или пересобранный контейнер используется для всех фаззинг-целей проекта. Каждая фаззинг-цель имеет индивидуальный корпус и отдельное задание фаззинга, которое последовательно запускает этапы конвейера анализа. Кроме того, существует возможность расширить начальный корпус входных данных файлами, полученными в ходе предыдущих сеансов фаззинга. Помимо журналов и статистики анализа, выходные данные задания фаззинга включают результирующий корпус, информацию о покрытии, отчеты Csgf и входные данные, вызывающие ошибки.

7. Экспериментальная оценка результатов

Мы постоянно используем sydr-fuzz для проведения программного анализа. Репозиторий OSS-Sydr-Fuzz [19] содержит список тестируемых проектов, инструкции по их сборке, настройки конфигурации для окружения и процесса фаззинга. За один год использования sydr-fuzz было обнаружено 85 ранее неизвестных дефектов в 22 проектах с открытым исходным кодом [51]. Из них 13 ошибок были найдены с помощью предикатов безопасности, реализованных в Sydr.

Для сравнения sydr-fuzz с современными инструментами фаззинга мы использовали фреймворк Google FuzzBench [22]. Эксперименты на FuzzBench развертывались на сервере

с 256 ГБ ОЗУ и двумя процессорами AMD EPYC 7702 (по 64 ядра каждый). Sydr-fuzz последовательно тестировался в сравнении с 4 фаззерами: libFuzzer [16], AFL++ [15], SymQEMU [9] и FUZZOLIC [10]. Мы выбрали 15 целей из FuzzBench, предназначенных для оценки покрытия. Эксперимент был настроен на проведение 10 прогонов для каждой комбинации фаззера и цели, каждый прогон выполнялся на одном ядре CPU в течение 23 часов. Из-за ограниченных ресурсов сервера мы разделили тестирование sydr-fuzz на две группы целей. Результаты наших экспериментов на FuzzBench общедоступны [23].

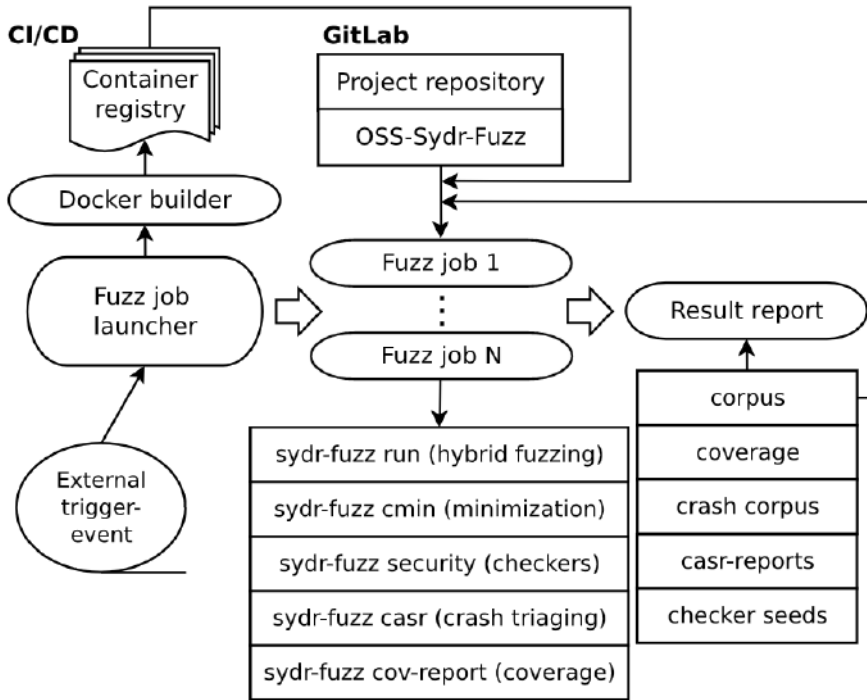


Рис. 1. CI-схема OSS-Sydr-Fuzz.
Fig. 1. CI-scheme for OSS-Sydr-Fuzz.

В ходе оценки использовалась следующая конфигурация sydr-fuzz. Гибридный фаззинг выполнялся с одним экземпляром фаззера и одним запущенным экземпляром Sydr в каждый момент времени. Sydr инвертировал ветви в прямом порядке с одним потоком решения. Ограничение в 10 секунд устанавливалось для решения одного SMT-запроса, и 60 секунд – для общего времени решения. Каждый запуск Sydr ограничивался 2 минутами. Использовался кэш для предотвращения инверсии одних и тех же ветвей Sydr. Для Sydr были включены строгое оптимистичное решение и фаззинг символических адресов. Каждый символический адрес обрабатывался фаззингом для до 10 различных моделей. Фаззинг символических адресов останавливался для текущего запуска Sydr, когда было сгенерировано 1000 таких моделей. Каждый 25-й запуск Sydr вместо фаззинга символических адресов включался режим полной обработки символических указателей. Процесс построения предиката пути приостанавливался после планирования 300 заданий на инверсию ветвей во время символического выполнения. Использование памяти для Sydr ограничивалось 8 ГБ. При превышении этого лимита выполнение программы завершается, и продолжается только инверсия ветвей.

7.1 Сравнение с фаззерами

Во-первых, мы сравнили sydr-fuzz с двумя современными фаззерами, чтобы доказать преимущества использования символического движка (рис. 2). Мы протестировали AFL++ [15] и libFuzzer [16] в сравнении с sydr-fuzz, сконфигурированного с соответствующим фаззером. Одинаковые версии libFuzzer (de5b16d) и AFL++ (8fc249d) использовались в sydr-fuzz и FuzzBench для оценки. Также во всех инструментах использовалась идентичная конфигурация фаззера. Поскольку в ходе одного испытания sydr-fuzz запускает два экземпляра (фаззер и Sydr) на одном CPU-ядре, мы сконфигурировали FuzzBench на запуск двух воркеров libFuzzer (-workers=2) и двух экземпляров AFL++ (основной и вторичный узлы).

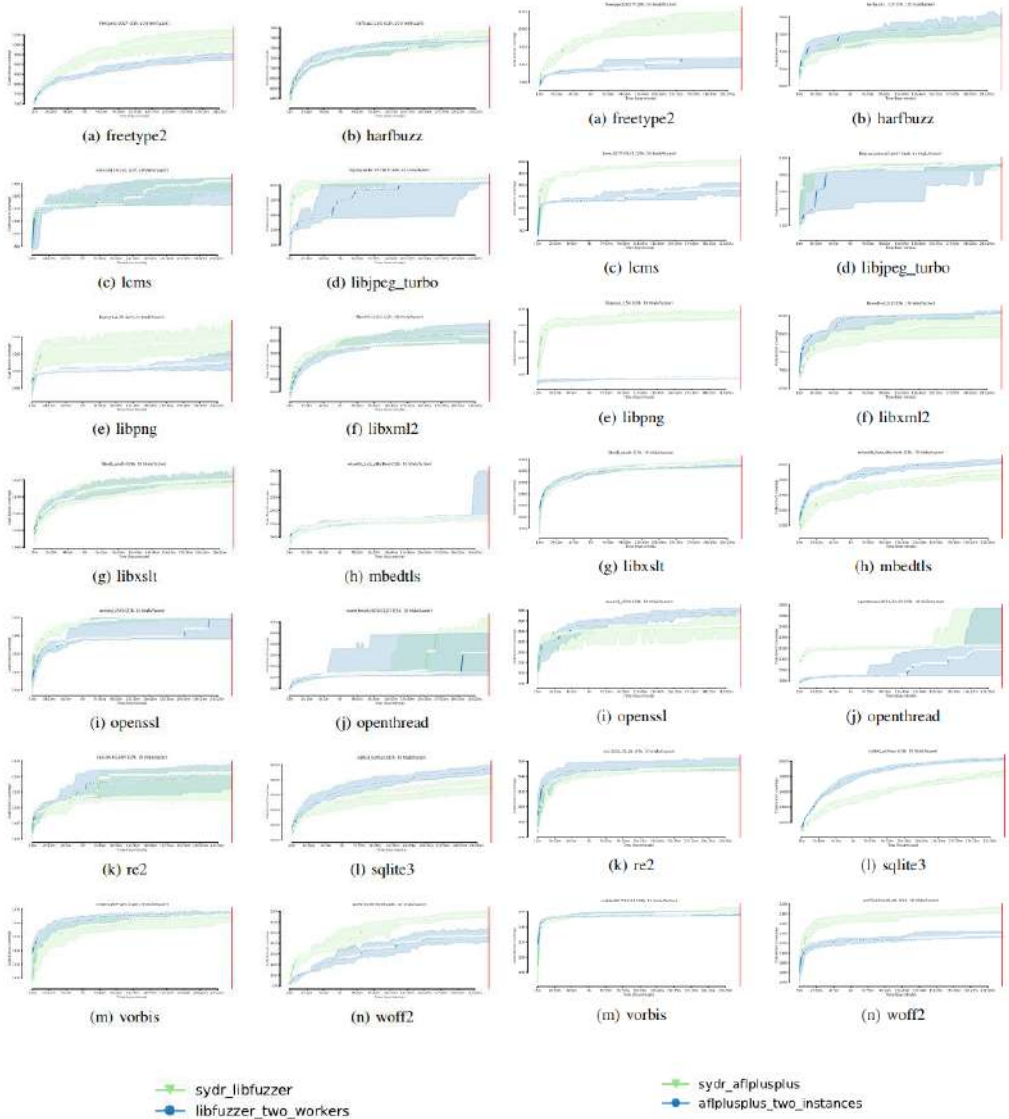


Рис. 2. Тестирование Sydr-Fuzz и фаззеров (234).
Fig. 2. Results of Sydr-Fuzz u fuzzer testing.

Результаты сравнения sydr-fuzz и libFuzzer показаны на рис. 2 в двух левых столбцах. Sydr-fuzz превзошел libFuzzer по достигнутому покрытию в 9 из 14 приложений. Из 5 приложений, где libFuzzer показал лучшие результаты, существенное преимущество наблюдается только для sqlite3. Для остальных 4 приложений итоговое покрытие незначительно отличается. Эксперимент проводился в двух запусках FuzzBench по 7 бенчмарков в каждом. Sydr-fuzz показал более высокий нормализованный показатель покрытия в обоих запусках: 98.67% и 99.63% для sydr-fuzz против 96.51% и 98.33% для libFuzzer соответственно.

В двух правых столбцах рис. 2 показаны результаты эксперимента sydr-fuzz в сравнении с AFL++. Sydr-fuzz также показал лучшие результаты в 9 из 14 приложений в этом эксперименте. Как видно из рис. 2, на большинстве приложений наблюдается значительный разброс результатов между sydr-fuzz и AFL++ в пользу обоих инструментов. Для двух групп целей sydr-fuzz получил более высокий средний показатель покрытия: 98.75% и 99.19% для sydr-fuzz против 94.87% и 96.70% для AFL++ соответственно.

7.2 Сравнение с гибридными фаззерами

На следующем этапе мы оценили sydr-fuzz в сравнении с современными гибридными фаззерами. Мы выбрали SymQEMU [9] и FUZZOLIC [10], поскольку эти инструменты также выполняют гибридный фаззинг на основе AFL++ [15] с символьным выполнением бинарного кода. Мы убедились, что sydr-fuzz использует ту же версию AFL++ (8fc249d) и настройки, что и эти инструменты. Набор бенчмарков незначительно отличается от использованного для оценки фаззеров из-за несовместимости тестируемых инструментов с некоторыми целями: оба инструмента не работали с libxslt и openssl, а FUZZOLIC – с woff2. Результаты сравнения sydr-fuzz и SymQEMU показаны на рис. 3 в двух левых столбцах.

Sydr-fuzz смог превзойти SymQEMU в 7 из 13 приложений. Результаты довольно близки на большинстве бенчмарков. Из всех бенчмарков, где SymQEMU показал лучшие результаты, только для zlib_uncompress наблюдается значительное преимущество над sydr-fuzz. Для бенчмарка sqlite3 было запущено только 5 испытаний из-за нестабильности SymQEMU на этой цели. Для двух групп экспериментов sydr-fuzz показал более высокое среднее покрытие: 99.35% и 99.95% для sydr-fuzz против 97.03% и 99.67% для SymQEMU соответственно.

Результаты экспериментов с sydr-fuzz и FUZZOLIC показаны в правых двух столбцах рис. 3. Для оценки FUZZOLIC было доступно только 12 целей. Sydr-fuzz смог превзойти FUZZOLIC в 6 из 12 бенчмарков. Как и в случае с SymQEMU, результаты в этом эксперименте близки. FUZZOLIC показал значительно большее покрытие только на цели sqlite3. Sydr-fuzz существенно превзошел FUZZOLIC на целях libjpeg_turbo и openthread. На остальных бенчмарках результаты очень схожи по истечении 23 часов. Тем не менее, sydr-fuzz смог достичь немного более высокого среднего нормализованного показателя покрытия в обеих группах экспериментов: 99.1% и 99.84% для sydr-fuzz против 99.07% и 99.81% для FUZZOLIC соответственно.

Таким образом, sydr-fuzz превзошел как libFuzzer, так и AFL++ на большинстве оцениваемых бенчмарков и достиг более высокого общего покрытия. Наши результаты также показывают, что sydr-fuzz находится на том же уровне, что и мощные современные гибридные фаззеры, и в некоторых случаях может их превосходить. Кроме того, sydr-fuzz смог превзойти все протестированные coverage-guided и гибридные фаззеры на 4 целях: freetype2, openthread, libxslt (SymQEMU и FUZZOLIC не смогли выполнить libxslt) и woff2 (FUZZOLIC не удалось запустить). Также есть одна цель, sqlite3, на которой все другие инструменты показали лучшие результаты, чем sydr-fuzz. Это можно объяснить неэффективной работой Sydr в данном конкретном случае, что снижает общую производительность фаззинга.

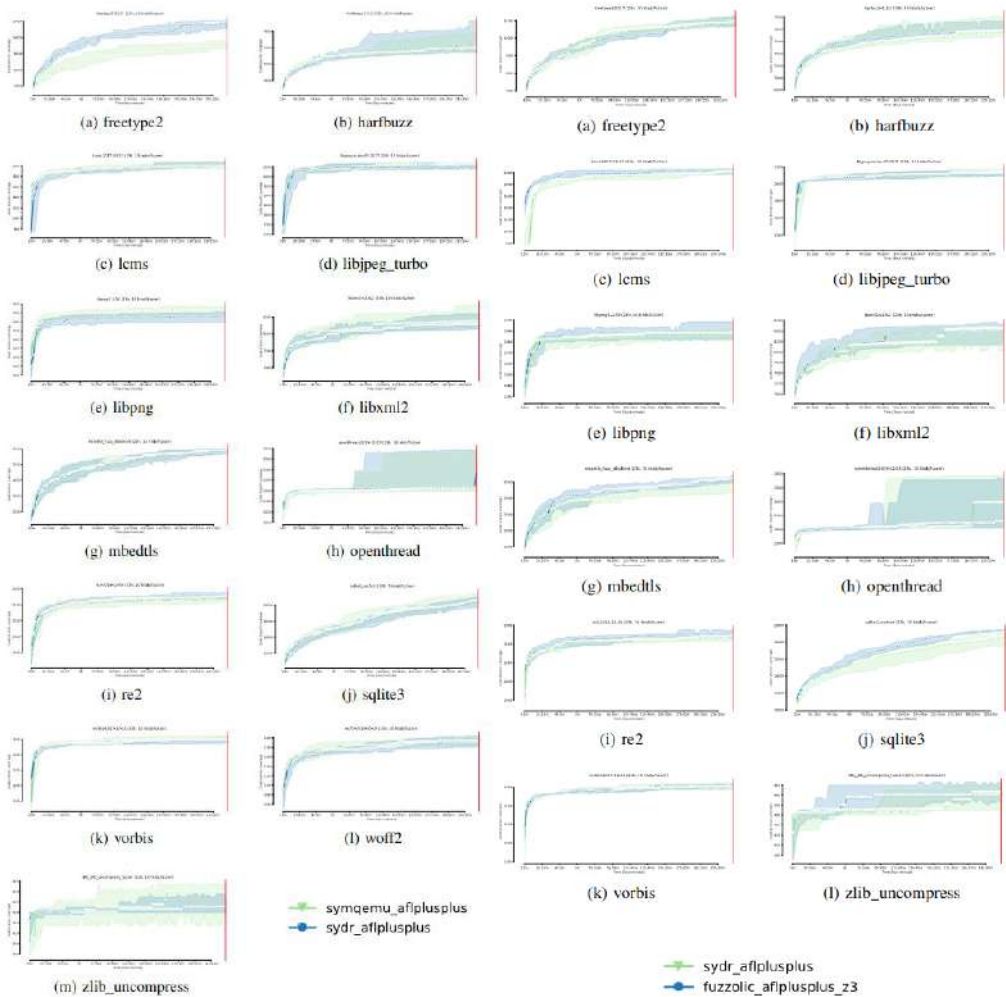


Рис. 3. Тестирование Sydr-Fuzz и гибридных фаззеров (23ч).
 Fig. 3. Results of Sydr-Fuzz and hybrid fuzzers testing.

8. Заключение

Мы представили фреймворк непрерывного гибридного фаззинга Sydr-Fuzz для эффективного динамического анализа программ в рамках жизненного цикла разработки безопасных приложений. Мы объединили оркестрацию гибридного фаззинга, минимизацию корпуса, обнаружение ошибок, сбор информации о покрытии и классификацию сбоев в единый набор инструментов. Мы представили новую интеграцию гибридного фаззинга на основе символьного исполнителя Sydr [14] и популярных фаззеров с открытым исходным кодом AFL++ [15] и libFuzzer [16]. Мы создали репозиторий OSS-Sydr-Fuzz [19] с целями из ПО с открытым исходным кодом для sydr-fuzz и обнаружили 85 новых ошибок в 22 проектах [51]. Мы предложили конвейер динамического анализа для sydr-fuzz, чтобы максимизировать полезный эффект от набора инструментов. Мы открыли для использования исходный код инструмента Casg для кластеризации и дедупликации отчетов о сбоях [24].

Наша оценка показывает, что, с одной стороны, sydr-fuzz превосходит современные фаззеры с покрытием AFL++ и libFuzzer на большинстве оцениваемых целей и достигает более высокого общего покрытия. С другой стороны, sydr-fuzz оказался сопоставим с мощными современными гибридными фаззерами SymQEMU [9] и FUZZOLIC [10] и в некоторых случаях даже может их превзойти. Значительная польза, которую sydr-fuzz получает от динамического символического исполнителя Sydr в процессе фаззинга, таким образом, демонстрирует актуальность использования передовых гибридных фаззеров в динамическом анализе.

Список литературы / References

- [1]. M. Howard and S. Lipner, The security development lifecycle. Microsoft Press Redmond, 2006, vol. 8. [Online]. Available: <http://msdn.microsoft.com/en-us/library/ms995349.aspx> (accessed 24.09.2025).
- [2]. ISO/IEC 15408-3:2008: Information technology – Security techniques – Evaluation criteria for IT security – Part 3: Security assurance components. 2008. [Online]. Available: <https://www.iso.org/standard/46413.html> (accessed 24.09.2025).
- [3]. GOST R 56939-2016: Information protection. Secure software development. General requirements. National Standard of Russian Federation, 2016. [Online]. Available: <http://protect.gost.ru/document.aspx?control=7&id=203548> (accessed 24.09.2025).
- [4]. S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, Unleashing Mayhem on binary code, in Proceedings of the 2012 IEEE Symposium on Security and Privacy, ser. SP '12, IEEE, 2012, pp. 380-394.
- [5]. B. S. Pak, Hybrid fuzz testing: Discovering software bugs via fuzzing and symbolic execution, M.S. thesis, School of Computer Science Carnegie Mellon University, 2012.
- [6]. N. Stephens, J. Grosen, C. Salls, A. Dutcher, R. Wang, J. Corbetta, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, Driller: Augmenting fuzzing through selective symbolic execution, in NDSS, vol. 16, 2016, pp. 1-16.
- [7]. I. Yun, S. Lee, M. Xu, Y. Jang, and T. Kim, QSYM: A practical concolic execution engine tailored for hybrid fuzzing, in 27th USENIX Security Symposium, 2018, pp. 745-761.
- [8]. S. Poeplau and A. Francillon, Symbolic execution with SymCC: Don't interpret, compile! In 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 181-198.
- [9]. S. Poeplau and A. Francillon, SymQEMU: Compilation-based symbolic execution for binaries, in Proceedings of the 2021 Network and Distributed System Security Symposium, 2021.
- [10]. L. Borzacchiello, E. Coppa, and C. Demetrescu, FUZZOLIC: Mixing fuzzing and concolic execution, Computers & Security, vol. 108, p. 102 368, 2021.
- [11]. J. Chen, W. Han, M. Yin, H. Zeng, C. Song, B. Lee, H. Yin, and I. Shin, SYMSAN: Time and space efficient concolic execution via dynamic data-flow analysis, in 31st USENIX Security Symposium (USENIX Security 22), USENIX Association, 2022, pp. 2531-2548.
- [12]. R. David, J. Salwan, and J. Bourroux, From source code to crash test-cases through software testing automation, Proc. of the 28th C&ESAR, p. 27, 2021.
- [13]. FuzzBench symbolic report, 2021. [Online]. Available: <https://www.fuzzbench.com/reports/experimental/2021-07-03-symbolic/index.html> (accessed 24.09.2025).
- [14]. A. Vishnyakov, A. Fedotov, D. Kuts, A. Novikov, D. Parygina, E. Kobrin, V. Logunova, P. Belecky, and S. Kurmangaleev, "Sydr: Cutting edge dynamic symbolic execution," in 2020 Ivannikov ISPRAS Open Conference (ISPRAS), IEEE, 2020, pp. 46-54.
- [15]. A. Fioraldi, D. Maier, H. Eißfeldt, and M. Heuse, "AFL++: Combining incremental steps of fuzzing research," in 14th USENIX Workshop on Offensive Technologies (WOOT 20), 2020.
- [16]. K. Serebryany, "Continuous fuzzing with libFuzzer and AddressSanitizer," in 2016 IEEE Cybersecurity Development (SecDev), IEEE, 2016, p. 157.
- [17]. G. Savidov and A. Fedotov, "Casr-Cluster: Crash clustering for linux applications," in 2021 Ivannikov ISPRAS Open Conference (ISPRAS), IEEE, 2021, pp. 47-51.
- [18]. A. Vishnyakov, V. Logunova, E. Kobrin, D. Kuts, D. Parygina, and A. Fedotov, "Symbolic security predicates: Hunt program weaknesses," in 2021 Ivannikov ISPRAS Open Conference, IEEE, 2021, pp. 76-85.

- [19]. OSS-Sydr-Fuzz: Hybrid fuzzing for open source software. [Online]. Available: <https://github.com/ispras/oss-sydr-fuzz> (accessed 24.09.2025).
- [20]. OSS-Fuzz: Continuous fuzzing for open source software. [Online]. Available: <https://github.com/google/oss-fuzz> (accessed 24.09.2025).
- [21]. D. Kuts, "Towards symbolic pointers reasoning in dynamic symbolic execution," in 2021 Ivannikov Memorial Workshop (IVMEM), IEEE, 2021, pp. 42-49.
- [22]. J. Metzman, L. Szekeres, L. Simon, R. Sprabery, and A. Arya, "FuzzBench: An open fuzzer benchmarking platform and service," in Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2021, pp. 1393-1403.
- [23]. FuzzBench results for Sydr-Fuzz. [Online]. Available: <https://sydr-fuzz.github.io/fuzzbench> (accessed 24.09.2025).
- [24]. CASR: Crash analysis and severity report. [Online]. Available: <https://github.com/ispras/casr> (accessed 24.09.2025).
- [25]. L. Borzacchiello, E. Coppa, and C. Demetrescu, "Fuzzing symbolic expressions," in 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE, 2021, pp. 711-722.
- [26]. R. Swiecki and F. Gröbert, Honggfuzz. [Online]. Available: <https://github.com/google/honggfuzz> (accessed 24.09.2025).
- [27]. F. Soudel and J. Salwan, "Triton: A dynamic symbolic execution framework," in Symposium sur la sécurité des technologies de l'information et des communications, ser. SSTIC, 2015, pp. 31-54.
- [28]. C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04), vol. 4, 2004, p. 75.
- [29]. DataFlowSanitizer design document, 2018. [Online]. Available: <https://clang.llvm.org/docs/DataFlowSanitizerDesign.html> (accessed 24.09.2025).
- [30]. P. Chen and H. Chen, "Angora: Efficient fuzzing by principled search," in 2018 IEEE Symposium on Security and Privacy (SP), IEEE, 2018, pp. 711-725.
- [31]. J. Chen, J. Wang, C. Song, and H. Yin, "JIGSAW: Efficient and scalable path constraints fuzzing," in 2022 IEEE Symposium on Security and Privacy (SP), IEEE, 2022, pp. 1531-1531.
- [32]. OneFuzz: A self-hosted fuzzing-as-a-service platform. [Online]. Available: <https://github.com/microsoft/onefuzz> (accessed 24.09.2025).
- [33]. Grizzly browser fuzzing framework. [Online]. Available: <https://github.com/MozillaSecurity/grizzly>.
- [34]. K. Serebryany, "OSS-Fuzz - Google's continuous fuzzing service for open source software," USENIX Association, 2017.
- [35]. Fuzzit. [Online]. Available: <https://github.com/fuzzitdev/fuzzit> (accessed 24.09.2025).
- [36]. cifuzz: Fuzz tests as easy as unit tests. [Online]. Available: <https://github.com/CodeIntelligenceTesting/cifuzz> (accessed 24.09.2025).
- [37]. S. Warkentin, Getting started using Mayhem with continuous integration, 2020. [Online]. Available: <https://www.brighttalk.com/webcast/17668/439580> (accessed 24.09.2025).
- [38]. Y. Chen, Y. Jiang, F. Ma, J. Liang, M. Wang, C. Zhou, X. Jiao, and Z. Su, "EnFuzz: Ensemble fuzzing with seed synchronization among diverse fuzzers," in 28th USENIX Security Symposium (USENIX Security 19), 2019, pp. 1967-1983.
- [39]. S. Österlund, E. Geretto, A. Jemmett, E. Güler, P. Görz, T. Holz, C. Giuffrida, and H. Bos, "Collabfuzz: A framework for collaborative fuzzing," in Proceedings of the 14th European Workshop on Systems Security, 2021, pp. 1-7.
- [40]. J. D. DeMott, R. J. Enbody, and W. F. Punch, "Towards an automatic exploit pipeline," in 2011 International Conference for Internet Technology and Secured Transactions, IEEE, 2011, pp. 323-329.
- [41]. OSS-Fuzz issue report tracker. [Online]. Available: <https://bugs.chromium.org/p/oss-fuzz/issues/list?q=status%3AWontFix%2CDuplicate%20-component%3AInfra&can=1> (accessed 24.09.2025).
- [42]. Google cloud platform. [Online]. Available: <https://github.com/GoogleCloudPlatform> (accessed 24.09.2025).
- [43]. M. Copeland, J. Soh, A. Puca, M. Manning, and D. Gollob, "Microsoft azure," New York, NY, USA: Apress, pp. 3-26, 2015.
- [44]. A crash course to Radamsa. [Online]. Available: <https://gitlab.com/akihe/radamsa> (accessed 24.09.2025).

- [45]. Coverage-guided fuzz testing in GitLab. [Online]. Available: https://docs.gitlab.com/ee/user/application_security/coverage_fuzzing/ (accessed 24.09.2025).
- [46]. D. Vyukov, syzbot: Automated kernel testing, 2018. [Online]. Available: https://lpc.events/event/2/contributions/237/attachments/61/71/syzbot_automated_kernel_testing.pdf (accessed 24.09.2025).
- [47]. D. Parygina, A. Vishnyakov, and A. Fedotov, "Strong optimistic solving for dynamic symbolic execution," in *Ivannikov Memorial Workshop (IVMEM)*, IEEE, 2022.
- [48]. A. Niemetz and M. Preiner, "Bitwuzla at the SMT-COMP 2020," *CoRR*, vol. abs/2006.01621, 2020. arXiv: 2006.01621. [Online]. Available: <https://arxiv.org/abs/2006.01621> (accessed 24.09.2025).
- [49]. T. Wang, T. Wei, Z. Lin, and W. Zou, "IntScope: Automatically detecting integer overflow vulnerability in x86 binary using symbolic execution," in *NDSS*, 2009.
- [50]. GDB 'exploitable' plugin. [Online]. Available: <https://github.com/jfoote/exploitable> (accessed 24.09.2025).
- [51]. Sydr-Fuzz trophy list. [Online]. Available: <https://github.com/ispras/oss-sydr-fuzz/blob/master/TROPHIES.md> (accessed 24.09.2025).
- [52]. GitLab: The one devops platform. [Online]. Available: <https://about.gitlab.com/> (accessed 24.09.2025).
- [53]. D. Bruening, "Efficient, transparent, and comprehensive runtime code manipulation," Ph.D. dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2004.

Информация об авторах / Information about authors

Алексей Вадимович ВИШНЯКОВ – кандидат физико-математических наук, закончил бакалавриат и магистратуру ВМК МГУ в 2020 году. Сфера научных интересов: компьютерная безопасность, жизненный цикл безопасной разработки (SDL), анализ бинарного кода, символьная интерпретация, фаззинг, автоматическое обнаружение ошибок и компиляторы.

Alexey Vadimovich VISHNYAKOV – Cand. Sci. (Phys.-Math.), obtained BSc degree and M.D. in the Faculty of Computational Mathematics and Cybernetics at Lomonosov Moscow State University. Research interests: computer security, security development lifecycle (SDL), binary analysis, symbolic execution, fuzzing, automatic error detection, and compilers.

Даниил Олегович КУЦ – кандидат технических наук. Сфера научных интересов: анализ бинарного кода, символьное выполнение, гибридный фаззинг.

Daniil Olegovich KUTS – Cand. Sci. (Phys.-Math.) since 2023. Research interests: binary analysis, symbolic execution, hybrid fuzzing.

Влада Игоревна ЛОГУНОВА – научный сотрудник лаборатории системного программирования и информационной безопасности Института системного программирования. Сфера научных интересов: динамический анализ, анализ бинарного кода, динамическая символьная интерпретация, гибридный фаззинг.

Vlada Igorevna LOGUNOVA – Research Fellow at the Laboratory of System Programming and Information Security of the Institute for System Programming. Research interests: dynamic analysis, binary code analysis, dynamic symbolic execution, hybrid fuzzing.

Дарья Алексеевна ПАРЫГИНА – магистр Московского государственного университета имени М.В. Ломоносова, старший лаборант Института системного программирования. Сфера научных интересов: символьное выполнение, гибридный фаззинг, направленный фаззинг.

Darya Alekseevna PARYGINA – master of Lomonosov Moscow State University, senior laborant of the Institute for System Programming of the RAS. Research interests: symbolic execution, hybrid fuzzing, directed fuzzing.

Илай Александрович КОБРИН – закончил бакалавриат и магистратуру ВМК МГУ в 2025 году. Сфера научных интересов: компьютерная безопасность, анализ бинарного кода, символьная интерпретация, фаззинг, операционные системы.

Eli Aleksandrovich KOBRIN obtained BSc degree and M.D. in the Faculty of Computational Mathematics and Cybernetics at Lomonosov Moscow State University. Research interests: computer security, binary analysis, symbolic execution, fuzzing, operating systems.

Георгий Анатольевич САВИДОВ – закончил бакалавриат и магистратуру ВМК МГУ в 2025 году. Сфера научных интересов: компьютерная безопасность, анализ бинарного кода, символьная интерпретация, фаззинг, операционные системы.

Georgy Anatolievich SAVIDOV obtained BSc degree and M.D. in the Faculty of Computational Mathematics and Cybernetics at Lomonosov Moscow State University. Research interests: computer security, binary analysis, symbolic execution, fuzzing, operating systems.

Андрей Николаевич ФЕДОТОВ – закончил НИЯУ МИФИ в 2013 году, кандидат технических наук с 2017 года. Сфера научных интересов: информационная безопасность, символьная интерпретация, оценка критичности ошибок, обратная инженерия, поиск ошибок, языки программирования, динамический анализ.

Andrey Nikolaevich FEDOTOV – Cand. Sci. (Tech.) since 2017, graduated from National Research Nuclear University МЕРФИ (Moscow Engineering Physics Institute) in 2013. Research interests: information security, symbolic execution, error severity estimation, reverse engineering, error search, programming languages, dynamic analysis.

