ТРУДЫ

ИНСТИТУТА СИСТЕМНОГО ПРОГРАММИРОВАНИЯ РАН

PROCEEDINGS OF THE INSTITUTE FOR SYSTEM PROGRAMMING OF THE RAS

ISSN Print 2079-8156
Том 37 Выпуск 5

ISSN Online 2220-6426
Volume 37 Issue 5

Институт системного
программирования
им. В.П. Иванникова РАН

Москва, 2025

ИСП РАН

ISSN 2079-8156 (Print), ISSN 2220-6426 (Online)

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:

УДК004.45

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/

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

Содержание

Table of Contents

DOI: 10.15514/ISPRAS-2025-37(5)-1

Кластеризация услуг распределённой сети, в которой хосты могут выполнять функцию коммутации сообщений

[1] *И. Б. Бурдонов,* ORCID: 0000-0001-9539-7853 <*igor@ispras.ru*>
[1,2] *Н. В. Евтушенко,* ORCID: 0000-0002-4006-1161 <*evtushenko@ispras.ru*>
[1] *А. С. Косачев,* ORCID: 0000-0001-5316-3813 <*kos@ispras.ru*>
[1] *В. Н. Пономаренко*, ORCID: 0009-0002-2387-2760 <*vera@ispras.ru*>

[1] *Институт системного программирования РАН им. В.П. Иванникова, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*

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

Аннотация. Статья является продолжением предыдущей статьи авторов, в которой строится абстрактная модель распределенной сети, содержащей только хосты и коммутаторы. Хосты предлагают пользователям пакеты определенных услуг (сервисов), сообщения (запросы) между хостами пересылаются через промежуточные узлы по правилам коммутации, и настройка узлов определяет множество путей от хоста к хосту, по которым будут пересылаться пакеты. Ситуация моделируется с использованием графа физических связей, вершинами которого являются хосты и коммутаторы, причем каждый хост (как и коммутатор) содержит систему правил коммутации. Обсуждается возможность повышения эффективности работы сети на основе использования информации о классах услуг, на которые разбивается множество всех услуг, предоставляемых хостами сети. На основе информации о классах услуг рассматриваются задачи передачи сообщений, настройка, в том числе инкрементальная, распределенной сети при различных изменениях параметров сети.

Ключевые слова: распределенная сеть; хосты; коммутаторы; классы услуг; (инкрементальная) настройка сети.

Для цитирования: Бурдонов И.Б., Евтушенко Н.В., Косачев А.С., Пономаренко В.Н. Кластеризация услуг распределённой сети, в которой хосты могут выполнять функцию коммутации сообщений. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 7–32. DOI: 10.15514/ISPRAS–2025–37(5)–1.

Clustering Services of Distributed Networks in Which Hosts Can Perform Message Switching Functions

[1] *I.B. Burdonov* ORCID: 0000-0001-9539-7853 <*igor@ispras.ru*>
[1,2] *N.V. Yevtushenko* ORCID: 0000-0002-4006-1161 <*evtushenko@ispras.ru*>
[1] *A.S. Kossatchev* ORCID: 0000-0001-5316-3813 <*kos@ispras.ru*>
[1] *V. N. Ponomarenko,* ORCID: 0009-0002-2387-2760 <*vera@ispras.ru*>

[1] *Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.*
[2] *National Research University Higher School of Economics, 20, Myasnitskaya st., Moscow, 101000, Russia.*

Abstract. The paper continues the previous work of the authors where an abstract model of a distributed network containing only hosts and switches has been developed. Hosts offer certain services to users; messages (requests) between hosts are forwarded through intermediate nodes according to switching rules, and the node configuration determines a set of paths from host to host along which packets are forwarded. The situation is modeled using a graph of physical connections where the graph nodes are hosts and switches, and each host (like a switch) contains a system of switching rules. The possibility of increasing the efficiency of the network is based on using of information about service classes, into which the set of all services provided by network hosts is divided. Based on the information about service classes, the tasks of message transmission, (incremental) node configuration are considered depending on various changes of network parameters.

Keywords: distributed network; hosts, switches; service classes; (incremental) network configuration.

For citation: Burdonov I.B., Yevtushenko N.V., Kossatchev A.S., Ponomarenko V.N. Clustering services of distributed networks in which hosts can perform message switching functions. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 7-32 (in Russian). DOI: 10.15514/ISPRAS-2025-37(5)-1.

1. Введение

Данная статья является продолжением статьи [1], в которой авторы строят абстрактную модель распределенной сети. Сеть содержит только хосты, которые предлагают пакеты определенных услуг (сервисов), и сообщения (запросы) между которыми пересылаются через промежуточные узлы (коммутаторы). Подобно тому, как это происходит в большинстве программно-конфигурируемых сетей [2-7] коммутатор работает по правилам коммутации, которые определяют, каким соседним узлам пересылается принятое коммутатором сообщение в зависимости от того, откуда оно пришло, и от вектора параметров в его заголовке. Как следствие, множество путей от хоста к хосту, по которым будут пересылаться пакеты, определяется настройкой коммутаторов. В качестве модели такой сети используется граф физических связей, вершинами которого являются хосты и коммутаторы, а ребра соответствуют физическим связям между ними. В общем случае хосты в такой сети принимают, обрабатывают и посылают информацию другим хостам, но в [1] мы предполагаем, что хост может выполнять также функции коммутации сообщений, т.е. содержит систему правил коммутации для пересылки полученного запроса/сообщения, если данный хост не может его обработать по каким-либо причинам. В программно-конфигурируемых сетях настройка правил коммутации обычно осуществляется специальными компонентами сети, например, SDN-контроллерами, однако в наших работах мы рассматриваем возможность самонастройки коммутаторов и хостов, в зависимости от передаваемых запросов, а также обсуждаем инкрементальную настройку при изменении параметров сети, в частности, её топологии.

В работе [1] обсуждаются проблемы, связанные с нефункциональными параметрами такой распределенной сети, а именно, *достижимость*/*недостижимость* хостов, *зацикливание* сообщений, *перегрузка* сети сообщениями, *немасштабируемость*, и возможности

оптимизации рассматриваемых параметров сети на основе использования информации об услугах/сервисах, предоставляемых каждым из хостов.

Данная статья является развитием идей и алгоритмов, предложенных в [1]. В частности, обсуждается возможность повышения эффективности работы сети на основе использования информации о классах услуг, на которые разбивается множество всех услуг, предоставляемых хостами сети. На основе информации о классах услуг рассматриваются задачи передачи сообщений, настройка, в том числе инкрементальная (повторная и частичная) настройка распределенной сети при различных изменениях параметров сети.

Структура работы следующая. В разделах 2 и 3 приводятся необходимые сведения из [1], касающиеся предлагаемой модели распределённой сети (раздел 2) и передачи по сети сообщений двух видов: сообщений указанному хосту-получателю и сообщений запроса услуг/сервисов, предоставляемых хостами с автоматическим выбором хоста, способного оказать запрашиваемую услугу (раздел 3). В последующих разделах обсуждается, как можно повысить эффективность работы сети с использования информации о классах услуг (раздел 4). Передача сообщений в сети с учётом классов услуг рассматривается в разделе 5, в разделе 6 обсуждается настройка распределенной сети, и соответственно, в разделе 7 — проблемы инкрементальной настройки функционирующей сети. В разделах 4-7 описаны идеи предлагаемых алгоритмов; сами алгоритмы (кроме тех, что приведены в [1] и сохраняются без изменений) вынесены в приложение.

2. Модель распределённой сети

В качестве модели распределённой сети (далее, просто сети) рассматривается связный неориентированный граф без кратных рёбер и петель, в котором вершины — это хосты и коммутаторы, а рёбра — каналы связи, по которым передаются сообщения. Ребру $\{a, b\}$ соответствуют две ориентированные дуги ab и ba, вершины a и b называются *соседними*.

В [1] предложена модель распределённой сети, в которой функцию коммутации сообщений выполняют как коммутаторы, так и хосты. Сообщение, принятое вершиной графа от соседней вершины, пересылается другому (или тому же самому) соседу вершины. Это определяется правилами коммутации, каждое из которых имеет вид $(p: a, s, b)$, где a, s и b вершины, причём a и b соседние с s вершины (в графе есть дуги as и sb), p — параметры коммутации как та часть параметров сообщения, которая определяет выбор того или иного правила. Если сообщение принято вершиной s от соседа a и параметры сообщения соответствуют параметрам коммутации p в правиле вида $(p: a, s, b)$, то это правило срабатывает, и сообщение пересылается соседу b. Будем называть соседа a *предшественником*, а соседа b *преемником*. Мы будем считать, что когда вершина s получает сообщение от предшественника a, идентификатор предшественника a является не параметром сообщения, а ответным параметром оператора приёма сообщения в s. Также когда вершина s посылает сообщение преемнику b, идентификатор преемника b является не параметром сообщения, а параметром оператора посылки сообщения.

В данной статье мы предполагаем, что сообщение не клонируется, то есть может сработать только одно правило — нет двух правил, отличающихся только получателем b.

Коммутатор выполняет только функцию коммутации сообщений, тогда как хост, кроме этого, генерирует сообщения, передаваемые далее по сети, и именно хост является конечным получателем сообщений.

Правила коммутации порождают пути в графе, по которым двигаются сообщения, сгенерированные хостами. Путь $a_1, a_2, ..., a_n$, где a_1 и a_n хосты, порождается правилами $(p: a_1, a_2, a_3)$, $(p: a_2, a_3, a_4)$, ..., $(p: a_{n-2}, a_{n-1}, a_n)$.

Сообщение может быть предназначено либо одному указанному хосту, либо «какому-нибудь» хосту, который может принять сообщение и обработать запрос, содержащийся в сообщении. В первом случае указывается идентификатор хоста-получателя, а во втором

случае — имя *услуги*, которую должен оказать хост-получатель хосту-отправителю. В первом случае сообщение принимается на обработку хостом, идентификатор которого указан в сообщении как идентификатор получателя, а во втором случае — каким-нибудь хостом, который может оказать запрашиваемую услугу, имя которой является параметром сообщения, такой хост в [1] назывался *целевым* хостом для данной услуги. Для этого в каждом хосте должно храниться множество имён услуг, которые этот хост может оказывать, т.е. реализованных в нём услуг. Если вершина, получившая сообщение, является коммутатором или хостом, который не может оказать запрашиваемую услугу, сообщение пересылается дальше согласно правилам коммутации.

Каждое сообщение содержит тип (имя) сообщения и набор параметров сообщения. Мы будем предполагать, что при передаче сообщения по сети может меняться только тип сообщения, а параметры сообщения передаются неизменными.

В [1] предложена установка таких правил коммутации в вершинах графа сети, которые порождают один (ориентированный) цикл, содержащий все хосты и называемый *циклом хостов*. Этот цикл строится как обход *дерева хостов*, которое определяется как поддерево остовного дерева графа с выделенным хостом-корнем (далее там, где это не приводит к двусмысленности, просто корнем), которое содержит все хосты, и все листовые вершины которого являются хостами (рис. 1). Заметим, что дерево хостов может содержать, кроме хостов, коммутаторы, необходимые для связности дерева. Дерево хостов строится процедурой «Удаление «лишних» коммутаторов». Первоначально дерево хостов совпадает с остовным деревом. Если коммутатор является листом дерева, он удаляется вместе с единственным инцидентным ему ребром дерева. Процедура повторяется до тех пор, пока все листья дерева не будут хостами. Полученное дерево и будет деревом хостов.

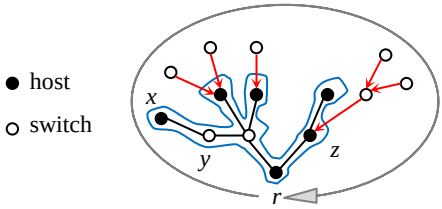


Рис. 1. Обход дерева хостов и лес деревьев коммутаторов.
Fig. 1. Host tree traversal and a forest of switch trees.

При обходе дерева хостов каждое ребро дерева хостов проходится ровно два раза, по одному разу в каждом направлении, т.е. цикл обхода является простым по дугам путём (не проходит дважды по одной дуге). Именно это свойство обеспечивает отсутствие клонирования сообщений. Для данного корневого дерева его обход, начиная с корня, однозначно определяется линейным порядком соседей вершины, заданным для каждой вершины: соседи вершины проходятся алгоритмом в этом порядке. *Родителем некорневой вершины x* будем называть такую вершину *y*, что дуга *xy* последняя на пути от корня до вершины *x*; при обходе по этой дуге мы первый раз попадаем в вершину *x*. *Родителем корня r* будем условно называть его последнего соседа *z* в линейном порядке соседей корня; при обходе дуга *zr* проходится последней.

За пределами дерева хостов оказывается часть дерева хостов, представляющая собой лес корневых поддеревьев остовного дерева, корни которых лежат на цикле хостов, а все остальные вершины являются коммутаторами. Будем называть эти поддеревья *деревьями коммутаторов*. Два дерева коммутаторов не имеют общих вершин. Для передачи сообщений по сети деревья коммутаторов не нужны, так как все хосты (а только они генерируют сообщения) находятся на цикле хостов, и сгенерированные сообщения двигаются по циклу хостов и не попадают в коммутаторы, не являющиеся корнями деревьев коммутаторов.

Однако лес деревьев коммутаторов может быть нужен, когда происходит изменение графа сети. Например, может добавляться новый хост, подсоединяемый к коммутатору на дереве коммутаторов. Поэтому будет считать, что правила коммутации порождают, кроме цикла хостов, лес деревьев коммутаторов, ориентированных к своим корням (на рис. 1 отмечены красным цветом).

В настроенной сети в каждой вершине инициализированы следующие переменные: *Self* — собственный идентификатор вершины, *Host* — отметка хоста, *Rules* — правила коммутации, *Root* — отметка корня дерева хостов, где «отметка» — булевская переменная.

3. Передача сообщений по циклу хостов

Сообщение с указанием идентификатора хоста-получателя имеет тип ***MessageToHost*** или ***RootMessageToHost***. Параметры сообщения: *sender* — идентификатор хоста-отправителя, *recipient* — идентификатор хоста-получателя, *parameters* — параметры сообщения, прозрачные для коммутации сообщений. Сообщение такого типа двигается по циклу хостов до хоста *recipient*, который и принимает сообщение на обработку, не пересылая его дальше. Для предотвращения бесконечного зацикливания сообщение, которое гарантированно прошло полный цикл хостов, удаляется. Поскольку цикл хост — это простой по дугам путь, каждая его дуга (но не обязательно вершина!) проходится один раз. Если сообщение ***MessageToHost*** проходит дугу от родителя корня в корень, не являющийся получателем сообщения, сообщение посылается дальше уже с типом ***RootMessageToHost***, а если по этой дуге проходит сообщение ***RootMessageToHost***, оно удаляется. Далее корень генерирует сообщение типа ***MessageToHost*** с отрицательным ответом хосту-отправителю. Последнее происходит, когда в сети нет хоста с требуемым идентификатором получателя. Тем самым, предотвращается бесконечная циркуляция по циклу хостов сообщений указанному хосту.

Схема передачи по сети сообщения известному получателю наглядно изображена на рис. 2, где двойная стрелка соответствует передаче сообщения по циклу хостов через вершины, не меняющие тип сообщения, цветной кружок означает хост-отправителя или хост-получатель, а белый кружок означает проход по дуге от родителя корня в корень. При передаче сообщений по этой схеме может случиться неправильное поведение: сообщение (ответное сообщение с отрицательным ответом от корня) может быть удалено, хотя в сети есть получатель этого сообщения (получатель ответного сообщения, т.е. отправитель исходного сообщения). Однако это может случиться только тогда, когда меняется корень, т.е. удаляется корень, сеть перенастраивается, и корнем становится другой хост. Удаление получателей и отправителей сообщения, отличных от корня, не приводит к такому некорректному поведению.

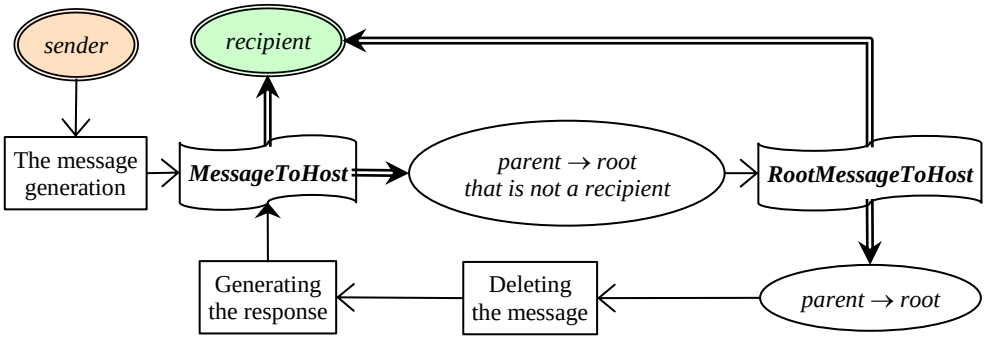


Рис. 2. Схема передачи по сети сообщения известному получателю.
Fig. 2. The network transmission diagram when sending a message to the known recipient.

Для генерации сообщения ***MessageToHost*** (***RootMessageToHost***) конкретному указанному хосту используется процедура ***SendMessageToHost*** с параметром *recipient* — идентификатор хоста, которому предназначено сообщение. Эта процедура посылает по циклу хостов сообщение ***MessageToHost***, если данный хост не корень дерева хостов, или сообщение ***RootMessageToHost*** в противном случае.

Сообщение с указанием имени запрашиваемой услуги может быть трёх типов: ***Message***, ***RootMessage*** или ***WaitingMessage***. Параметры сообщения: *sender* — идентификатор хоста-отправителя, *service* — имя запрашиваемой услуги, *parameters* — параметры сообщения, прозрачные для коммутации сообщений.

Сначала сообщение, как правило, посылается с типом ***Message***. Если оно проходит по дуге от родителя корня в корень, который не является целевым хостом, сообщение пересылается дальше по циклу с типом ***RootMessage***. Если сообщение ***Message*** или ***RootMessage*** приходит в целевой хост, но сейчас он «занят», хост пересылает сообщение дальше уже с типом ***WaitingMessage***. Корень, не являющийся целевым хостом, получая по дуге, ведущей от родителя корня в корень, сообщение ***WaitingMessage***, пересылает его дальше опять с типом ***Message***. Последнее сделано для того, что «ловить» удаление из цикла хостов того занятого целевого хоста, который сменил тип сообщения с ***Message*** на ***WaitingMessage***. Если в цикле ещё остаются целевые хосты, сообщение, проходя через целевой хост, либо принимается им, если хост «свободен», либо посылается дальше опять с типом ***WaitingMessage***, если хост сейчас «занят». Если сообщение ***RootMessage*** проходит по дуге, ведущей от родителя корня в корень, оно удаляется, а корень генерирует и посылает сообщение типа ***MessageToHost*** хосту-отправителю с отрицательным ответом.

Тем самым, в цикле может «крутится» только сообщение с запросом услуги, для которой в цикле хостов есть целевые хосты. Но это будет не бесконечно, а до тех пор, пока какой-нибудь целевой хост не освободится и не примет это сообщение, или пока из цикла хостов не будут удалены все целевые хосты. В то же время число проходов таким сообщением по циклу хостов не ограничено.

Схема передачи по сети сообщения с запросом услуги наглядно изображена на рис. 3, где двойная стрелка соответствует передаче сообщения по циклу хостов через вершины, не являющиеся целевыми хостами или корнем, цветной кружок означает хост-отправитель или целевой хост, а белый кружок означает проход по дуге от родителя корня в корень. Многоточие показывает, что передача по сети отрицательного ответа от корня отправителю исходного сообщения выполняется, естественно, по общим правилам для сообщения типа ***MessageToHost*** (как на рис. 2).

Предполагается, что когда хост принимает сообщение, оказывая запрашиваемую услугу, то после этого он может, если нужно, сам послать ответ хосту-отправителю, сгенерировавшему это сообщение с результатами оказания услуги. Заметим, что это не всегда нужно. Например, когда требуется некоторая цепочка услуг или более сложная программа, отдельные части которой понимаются как услуги, которые могут быть реализованы в разных хостах, то ответ, если нужно, будет послан хосту-отправителю только после выполнения всей программы. В любом случае сообщение-ответ посылается с типом ***MessageToHost*** или (из корня) с типом ***RootMessageToHost***. В сообщении в качестве получателя указывается отправитель исходного сообщения, ответ на которое посылается.

Для генерации в хосте сообщения запроса удалённой услуги используется процедура ***SendMessage*** с параметрами: *service* — имя запрашиваемой услуги, *parameters* — параметры сообщения, прозрачные для коммутации сообщений. Процедура возвращает ***false***, если некому послать сообщение с запросом услуги, т.е. в хосте нет правил коммутации (хост изолированная вершина). В противном случае возвращается ***true*** и посылается нужное сообщение запроса услуги: ***Message***, если отправитель не является целевым хостом или корнем, ***RootMessage***, если отправитель является корнем, но не является целевым хостом,

WaitingMessage, если отправитель является целевым хостом. В последнем случае предполагается, что хост делает удалённый вызов потому, что сейчас «занят».

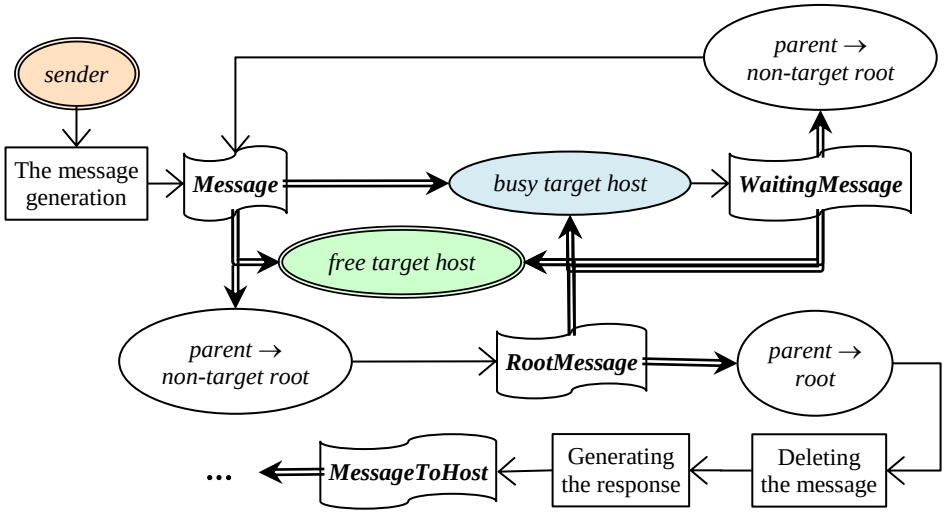


Рис. 3. Схема передачи по сети сообщения запроса услуги.
Fig. 3. The network transmission diagram when sending a service request message.

4. Классы услуг

У решения, предложенного в [1] и кратко описанного в предыдущем разделе, есть один существенный недостаток: путь, который должно пройти сообщение, прежде чем найдёт нужный хост, может оказаться слишком длинным. Например, если в сети запрашиваемая услуга реализована только в одном хосте *a*, то сообщение с этим именем услуги, сгенерированное хостом *b*, следующим после хоста *a* в цикле хостов, пройдёт весь цикл хостов, кроме коммутаторов между *a* и *b*.

Для того чтобы устранить этот недостаток, можно было бы строить для каждой услуги обход не дерева хостов, а минимального дерева, содержащего, быть может, не все хосты, но все целевые хосты, т.е. хосты, в которых эта услуга реализована. Сообщение с запросом данной услуги двигалось бы не циклу хостов, а по циклу обхода такого минимального дерева.

Однако в этом случае имя услуги становится частью параметров коммутации. Поэтому всё зависит от того, сколь велико число услуг, реализуемых сетью, и имеет ли оно тенденцию к росту. Существуют сети с фиксированным множеством реализуемых услуг, однако могут быть и другие сети, которые предоставляют пользователям, вообще говоря, нефиксированный набор услуг, который к тому же может расти вместе с развитием сети. Например, облачные вычисления имеют такую характеристику, как эластичность — услуги могут быть предоставлены, расширены, сужены в любой момент времени. Для таких сетей предлагаемое решение с циклом для каждой услуги приведёт к слишком большому числу правил коммутации в каждом узле сети, к тому же имеющим тенденцию к росту при росте числа услуг, что можно понимать как немасштабируемость сети, зависимой от такого параметра как число услуг.

Для решения такой проблемы немасштабируемости мы предлагаем воспользоваться некоторой кластеризацией услуг. В этом случае множество всех услуг разбивается на подмножества, которые будем называть *классами услуг*. В сообщении указывается не только имя услуги, но также имя её класса, вычисляемого хостом-отправителем по заданному отображению имени услуги в имя её класса. Коммутация сообщения происходит по имени

класса услуги, т.е. имя класса становится параметром коммутации, правило коммутации сообщений с запросом услуг имеет вид (*class*: *a, b, c*). Хост принимает сообщение на обработку по имени услуги, если она в нём реализована и он сейчас не «занят». Предполагается, что число классов услуг мало, а число услуг велико, новые классы услуг появляются редко, тогда как новые услуги в классах могут появляться чаще, а кроме того, в сети число хостов, реализующих услуги данного класса, как правило, ощутимо меньше общего числа хостов.

Примером могут служить различные приложения, к которым можно получить онлайн-доступ через интернет. Класс арифметических вычислений (функции сложения, вычитания, умножения, деления и т.п.) выполняют различные онлайн-калькуляторы, например, *https://calculator888.ru/* или *https://okcalc.com/ru/*. Последний калькулятор может вычислять также логарифм, но эту функцию может выполнять и специальный калькулятор логарифма *https://umath.ru/calc/vychislenie-logarifma-chisla-onlajn/*. Предел функции вычисляется приложением *https://mathdf.com/lim/ru/*, но также входит в число функций, реализуемых приложением *https://mathsolver.microsoft.com/ru/algebra-calculator*, который, кроме этого, умеет решать линейные и квадратные уравнения, неравенства, вычислять производные и интегралы. Последние вычисляются и специальным калькулятором интегралов *https://www.integral-calculator.ru/*. График функции можно построить с помощью приложения *https://yotx.ru/*, это умеет делать и графический калькулятор Desmos (*https://www.desmos.com/?lang=ru*), но у него гораздо больше возможностей (реализованных функций). И так далее.

Мы привели примеры услуг и их группировки по разным приложениям. В то же время такие приложения — не то же самое, что наши классы услуг. Группировка услуг в приложениях выполняется по самым разным критериям, отличающимся в разных приложениях. Кроме того, такая группировка — это не разбиение (как у нас на классы), а покрытие множества услуг, поскольку одна и та же услуга может быть реализована в разных приложениях. Наше разбиение услуг на классы можно сделать устойчивым к изменениям множества услуг и, тем более, к их реализации и группировке в различных приложениях. Например, класс услуг по вычислению элементарных функций может включать арифметические действия, степенную, показательную, логарифмическую функции и тригонометрические и обратные тригонометрические функции, и не включать другие функции. Этот класс услуг не зависит от того, в каких приложениях реализованы элементарные функции (услуги) и есть ли приложение, реализующее в точности этот набор функций (услуг).

Далее будем считать, что разбиение множества услуг на классы задано. Теперь *целевой хост* будем определять не для услуги, а для класса услуг — это хост, реализующий хотя бы одну услугу этого класса. Мы будем строить *дерево целевых хостов* и *цикл целевых хостов* для каждого класса услуг. Дерево целевых хостов определяется как поддерево дерева хостов с корнем в целевом хосте, содержащее все целевые хосты, все листовые вершины которого являются целевыми хостами. Заметим, что дерево целевых хостов может содержать, кроме целевых хостов, коммутаторы и нецелевые хосты, необходимые для связности дерева. Цикл целевых хостов определяется как цикл обхода дерева целевых хостов, этот цикл порождается правилами коммутации. Дерево целевых хостов строится процедурой «Удаление «лишних» коммутаторов и нецелевых хостов», аналогичной процедуре «Удаление «лишних» коммутаторов» при построении дерева хостов, только вместо «лишних» коммутаторов в ней будут «лишние» коммутаторы и нецелевые хосты.

За пределами цикла целевых хостов для данного класса услуг могут остаться «лишние» нецелевые хосты. Если такой хост генерирует сообщение, запрашивающее услугу данного класса, то возникает вопрос, как сообщение попадёт на цикл целевых хостов? Для этого достаточно, чтобы правила коммутации порождали простые по дугам пути, которые начинаются в нецелевых хостах и заканчиваются в вершинах цикла целевых хостов, и вместе

с циклом целевых хостов суммарно содержат все хосты. Для того чтобы не было клонирования, нужно, чтобы эти пути не разветвлялись после прохода по общей дуге.

В [1] за пределами цикла хостов могли быть только коммутаторы, и на них строился лес деревьев коммутаторов, корни которых лежали на дереве хостов (рис. 1). Аналогично теперь для каждого класса услуг будем строить лес *деревьев нецелевых хостов*, который получается из остовного дерева удалением рёбер дерева целевых хостов и образующихся после такого удаления изолированных вершин. Дерево нецелевых хостов ориентировано к корню, лежащему на дереве целевых хостов, и все его некорневые вершины являются коммутаторами или нецелевыми хостами (рис. 4). В дерево нецелевых хостов входят коммутаторы, которые нужны, прежде всего, для связности дерева, но, кроме того, мы помещаем в дерево нецелевых хостов коммутаторы, не лежащие на путях по дереву из нецелевых хостов к корню, а именно, коммутаторы на деревьях коммутаторов (в том числе, терминальные коммутаторы), которые «лишние» для передачи сообщений по сети. Это делается по той же причине, по которой в [1] строились деревья коммутаторов, в которых вообще нет хостов. Такие деревья нужны для инкрементальной настройки сети [1], когда меняется топология сети (граф). Например, в сеть добавляется новый хост, соединяемый новыми рёбрами со «старыми» вершинами. Такими «старыми» вершинами могут быть эти «лишние» коммутаторы (некорневые коммутаторы деревьев коммутаторов), и они перестают быть «лишними».

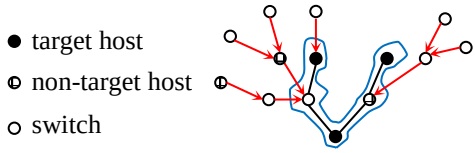


Рис. 4. Обход дерева целевых хостов и лес деревьев нецелевых хостов.
Fig. 4. Target host tree traversal and a forest of non-target host trees.

Для каждого класса услуг *class* в каждой вершине создаются свои «копии» переменных *Rules* и *Root*: *ClassToRules*(*class*) — правила коммутации для класса *class*, *ClassToRoot*(*class*) — отметка корня дерева целевых хостов для класса *class*.

5. Передача сообщений при наличии классов услуг

Сообщения с указанием идентификатора хоста-получателя типа ***MessageToHost*** или ***RootMessageToHost*** имеют те же параметры и передаются так же, как описано в разделе 3 и изображено на рис. 2, — по циклу хостов.

Сообщения с запросом услуги имеют те же три типа: ***Message***, ***RootMessage*** или ***WaitingMessage***. В параметры сообщения добавляется имя класса услуги: *sender* — идентификатор хоста-отправителя, *class* — имя класса услуги, *service* — имя запрашиваемой услуги, *parameters* — параметры сообщения, прозрачные для коммутации сообщений. Коммутация сообщений происходит по имени класса услуги, т.е. имя класса входит в параметры коммутации, правило коммутации сообщений с запросом услуг имеет вид (*class*: *a*, *b*, *c*).

Если сообщение с запросом услуги не находится на цикле целевых хостов (генерируется в нецелевом хосте на дереве нецелевых хостов и отлично от его корня), то сначала оно двигается по дереву нецелевых хостов к его корню, лежащему на цикле целевых хостов. Далее сообщение двигается (возможно, меняя свой тип) точно так же, как описано в разделе 3 и изображено на рис. 3, со следующими отличиями: 1) сообщение двигается не по циклу хостов, а по циклу целевых хостов для указанного класса услуг, 2) не всякий «свободный» целевой хост становится получателем сообщения, а только тот, в котором реализована

запрашиваемая услуга, 3) не всякий «занятый» целевой хост передаёт сообщение дальше под именем ***WaitingMessage***, а только тот, в ком реализована запрашиваемая услуга.

Для генерации сообщения запроса услуги используется процедура ***SendMessage***, как описано в разделе 3, с теми же параметрами. Отличие в том, что в сообщение добавляется имя класса запрашиваемой услуги, которое процедура вычисляет по имени услуги. Процедура возвращает ***false***, если некому послать сообщение с запросом услуги, т.е. в хосте нет правил коммутации для данного класса услуг: *ClassToRules*(*class*) = (). В противном случае возвращается ***true*** и посылается нужное сообщение запроса услуги с указанием её класса: ***Message*** — если в хосте не реализована запрашиваемая услуга, и хост не является корнем дерева целевых хостов, ***RootMessage*** — если в хосте не реализована запрашиваемая услуга, и хост является корнем дерева целевых хостов, ***WaitingMessage*** — если в хосте реализована запрашиваемая услуга, но сейчас хост «занят».

6. Настройка сети

Под настройкой сети понимается, прежде всего, установление правил коммутации. Например, программно-конфигурируемая сеть (SDN) основана на физическом разделении плоскости данных (уровень передачи сообщений, моделируемый графом с вершинами в хостах и коммутаторах) и плоскости управления сетью. На плоскости управления находится контроллер, который и выполняет настройку сети. Он связан с каждым коммутатором, которому «спускает» на уровень плоскости данных правила коммутации. Для этого на уровне контроллера должна быть известна вся топология сети (граф физических связей).

В настоящей статье, как и в [1], мы предлагаем алгоритмы самонастройки сети, без использования специального контроллера и с минимально необходимой информацией, заранее заданной в вершинах графа. Настройка разделяется на *первичную настройку* и *настройку для данного класса услуг*.

6.1 Первичная настройка сети

Первичная настройка, описанная в [1], определяет правила коммутации, порождающие цикл хостов и лес деревьев коммутаторов, а также отмечает корень дерева хостов. В результате первичной настройки в каждой вершине инициализируется список *Rules* правил коммутации и переменная *Root* := ***true*** в корне дерева хостов, *Root* := ***false*** в остальных хостах сети.

Все правила коммутации в вершине s имеют вид $(: a, s, b)$, т.е. параметры коммутации отсутствуют, а средняя вершина одна и та же — s. Поэтому в алгоритмах для краткости в каждой вершине s правило коммутации записывается не как $(: a, s, b)$, а как (a, b).

В каждой вершине s цикла хостов список *Rules* имеет «циклический вид» (a_0, a_1), (a_1, a_2), ..., (a_{n-2}, a_{n-1}), (a_{n-1}, a_0), где a_0 родитель вершины s.

Мы также вводим *правило умолчания*: если сообщение в вершину s приходит от соседа b, отличного от вершин a_0, ..., a_{n-1}, оно пересылается вершине a_0 — родителю вершины s по подразумеваемому правилу (b, a_0). По сути, это *правило умолчания*: список (a_0, a_1), (a_1, a_2), ..., (a_{n-1}, a_0) понимается как сокращённая запись списка (a_0, a_1), (a_1, a_2), ..., (a_{n-1}, a_0), (b_1, a_0), (b_2, a_0), ..., (b_k, a_0), где b_1, ..., b_k все соседи вершины s, кроме соседей a_0, ..., a_{n-1}. Такое *правило умолчания* мы применяем для упрощения алгоритмов самонастройки сети в Приложении. Оно используется в трёх случаях: 1) при инкрементальной (повторной частичной) перенастройке сети, когда меняется граф сети; 2) при передаче сообщений по дереву нецелевых хостов в настроенной сети; 3) при посылке сообщения в сеть извне сети, т.е. при посылке сообщения в вершину b с указанием идентификатора предшественника a, не совпадающего с идентификаторами вершин сети, что можно понимать как посылку сообщения по дуге ab, не входящей в граф сети.

На рис. 5 показано, какие правила создаются для вершины в разных случаях. Чёрные линии изображают рёбра дерева хостов, красные — деревьев коммутаторов. Правило (a, b) в

вершине *s* показано стрелкой, соединяющей две смежные дуги *as* и *sb*: синие стрелки соответствуют циклу хостов, а красные стрелки — деревьям коммутаторов. Показаны правила как до (рис. 5 сверху), так и после применения правила умолчания (рис. 5 внизу).

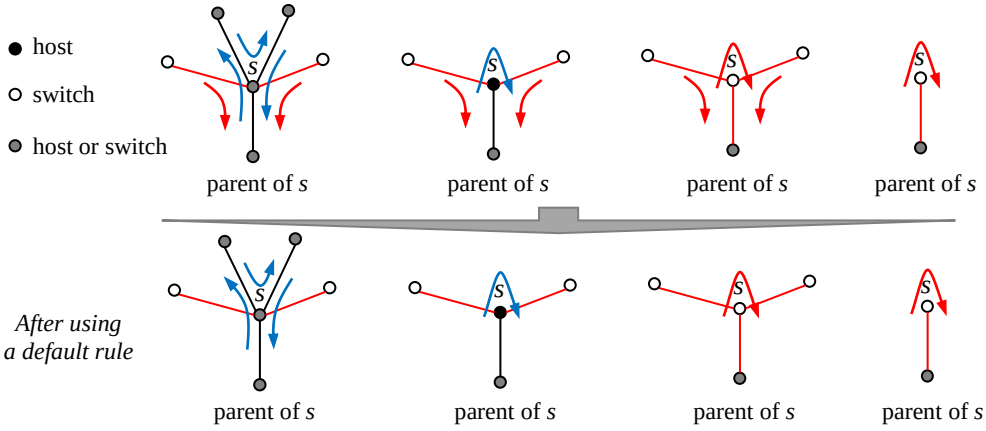


Рис. 5. Правила коммутации и правило умолчания.
Fig. 5. Switching rules and a default rule.

6.2 Настройка сети для заданного класса услуг

В данной статье мы вводим классы услуг, и для каждого класса услуг требуется своя дополнительная настройка сети, которая выполняется в предположении, что ранее была выполнена первичная настройка. Настройки для разных классов услуг выполняются независимо друг от друга, в том числе, они могут выполняться параллельно.

Настройка для заданного класса услуг определяет правила коммутации для этого класса услуг, порождающие цикл целевых хостов и лес деревьев нецелевых хостов.

В результате настройки для класса услуг *class* в каждой вершине инициализируется список *ClassToRules*(*class*) правил коммутации для этого класса услуг и переменная *ClassToRoot*(*class*) := ***true*** в корне дерева целевых хостов для этого класса услуг, *ClassToRoot*(*class*) := ***false*** в остальных хостах сети. Кроме того, каждой вершине сообщается множество *serviceSet* имён услуг данного класса услуг *class*, что позволяет инициализировать отображение *ServiceToClass* имени услуги в её класс для данного класса услуг: $\forall$ *service* $\in$ *serviceSet ServiceToClass*(*service*) := *class*.

Правило умолчания применяется и для правил коммутации для каждого класса услуг. Если на рис. 5 чёрный кружок понимать как целевой хост, белый кружок как коммутатор или нецелевой хост, а серый кружок как коммутатор или (любой) хост, то изображения на рисунке иллюстрируют правила умолчания для класса услуг.

Алгоритм настройки для данного класса услуг аналогичен алгоритму первичной настройки. Отличие в том, что при первичной настройке обходится весь граф, используя заранее инициализированный список соседей в каждой вершине, и строятся цикл хостов как обход дерева хостов и лес деревьев коммутаторов, а при настройке для заданного класса услуг список соседей не используется. В этом случае с использованием цикла хостов выполняется обход уже построенного дерева хостов, что определяется правилами коммутации, и строятся цикл целевых хостов как обход дерева целевых хостов и лес деревьев нецелевых хостов.

Настройка для данного класса услуг инициируется сообщением ***ClassStart***, которое поступает в некоторый (произвольный) хост извне графа. Этот хост будем называть *инициатором* для данного класса услуг. Для удобства будем считать, что сообщение приходит по некоторому дополнительному ребру, соединяющему инициатор с внешним окружением, моделируемым

дополнительной вершиной, соседней с инициатором, которую будем называть *внешним соседом* инициатора. Этому же внешнему соседу инициатор посылает ответ о выполнении настройки. Во время настройки в сети циркулирует одно сообщение, которое имеет тип (первоначально ***ClassStart***) и параметры: *class* — имя класса услуг, *serviceSet* — множество имён услуг этого класса.

Настройка проходит в три этапа: ***A***, ***B*** и ***C***.

На этапе ***A*** хост, получающий сообщение ***ClassStart***, становится инициатором, и организует поиск целевого хоста. Для этого сообщение с типом ***SearchA*** двигается по циклу хостов до целевого хоста или, если такого хоста нет, возвращается в инициатор. По ходу дела происходит инициализация в каждой вершине переменной *ClassToRules*(*class*) := () (пока целевой хост не найден, цикла целевых хостов нет), и инициализация в каждом хосте переменных *ClassToRoot*(*class*) := ***false*** и *ServiceToClass*(*class*) как описано выше. Если в сети нет целевого хоста, так будут проинициализированы переменные во всех вершинах. Тогда инициатор посылает своему внешнему соседу отрицательный ответ ***CancelC***. Если целевой хост найден, он становится корнем дерева целевых хостов (*ClassToRoot*(*class*) := ***true***) и начинается этап ***B***.

На этапе ***B*** выполняется обход дерева хостов с выполнением процедуры «Удаление «лишних» коммутаторов и нецелевых хостов», описанной в разделе 4. Тем самым строится цикл целевых хостов как обход дерева целевых хостов и лес деревьев нецелевых хостов. Заметим, что часть этого леса, а именно деревья коммутаторов, построена при первичной настройке и при настройке для данного класса услуг не меняется. Попутно происходит инициализация в каждой вершине переменной *ClassToRules*(*class*) := *Rules* (первоначально цикл целевых хостов совпадает с циклом хостов) и в каждом хосте переменных *ClassToRoot*(*class*) := ***false*** и *ServiceToClass*(*class*) как описано выше. Последние две инициализации, если целевой хост найден, могут быть уже выполнены в некоторых хостах на этапе ***A***, но не во всех хостах.

На этапе ***B*** сообщение может иметь один из трёх типов: *прямое* сообщение ***ForwardB*** и два *ответных сообщения* (*ответа*) ***CancelB*** и ***BackB***. Сообщение ***ForwardB*** посылается по дуге дерева хостов, ориентированного от корня. Когда это сообщение первый раз попадает в некоторую вершину *s*, пройдя по дуге *xs*, в вершине *s* сначала устанавливаются правила коммутации для данного класса такие же, как для цикла хостов, однако начало отсчёта циклического списка правил может стать другим, поскольку для дерева целевых хостов родителем вершины *s* считается вершина *x*. Если для цикла хостов в вершине *s* был список правил *Rules* = (a_0, a_1), (a_1, a_2), ..., (a_{n-2}, a_{n-1}), (a_{n-1}, a_0), где a_0 родитель вершины *s* в дереве хостов, то для $x = a_i$ в дереве целевых хостов сначала будет *ClassToRules*(*class*) = (a_i, a_{i+1}), (a_{i+1}, a_{i+2}), ..., (a_{n-2}, a_{n-1}), (a_{n-1}, a_0), (a_0, a_1), (a_1, a_2), ..., (a_{i-1}, a_i), где $x = a_i$ родитель вершины *s* в дереве целевых хостов. Это объясняется тем, что в общем случае корень дерева хостов может не быть целевым хостом для данного класса услуг. Смена корня при переходе от дерева хостов к дереву целевых хостов приводит, прежде всего, к изменению ориентации рёбер, когда деревья ориентированы от их корней. Это показано на рис. 6, где чёрные стрелки изображают совпадающую ориентацию рёбер в дереве хостов и дереве целевых хостов с корнем в вершине *x*, а красные стрелки показывают ориентацию рёбер в дереве целевых хостов, которая противоположна их ориентации в дереве хостов; оба дерева ориентированы от их корней. Там, где ориентация рёбер меняется, у вершины меняется её вершина-родитель: для дуги *ab* вершина *a* является родителем вершины *b*, поэтому у циклического списка правил в вершине выбирается, быть может, другое начальное правило. В алгоритмах в Приложении это изменение делается для каждой вершины *s*, хотя его достаточно сделать для вершин на пути по дереву хостов от «старого» до нового корня *x*, поскольку в остальных вершинах начальное правило списка правил не меняется, что видно по рис. 6.

Когда все выходящие из вершины *b* дуги дерева хостов, кроме дуги, ведущей к её родителю *a*, уже пройдены, вершина *b* посылает своему родителю *a* ответное сообщение. Это ответное

сообщение имеет тип ***BackB***, если поддерево дерева хостов с корнем в вершине *b* содержит целевой хост, или, в противном случае, тип ***CancelB***. В этом поддереве есть целевые хосты, если вершина *b* является целевым хостом, а также, если либо в вершине *b* определено больше одного правила коммутации (при использовании правила умолчания, рис. 5 внизу), либо в этих правилах есть хотя бы два разных преемника (без использования правила умолчания, рис. 5 вверху). Если посылается ответ ***BackB***, вершина *b* войдёт в дерево целевых хостов, а если посылается ответ ***CancelB***, вершина *b* войдёт в дерево нецелевых хостов. Тем самым, будет выполняться процедура «Удаление «лишних» коммутаторов и нецелевых хостов». Соответствующим образом корректируются правила коммутации для данного класса услуг. Этап ***B*** заканчивается, когда пройден весь цикл хостов, тогда начинается этап ***C***.

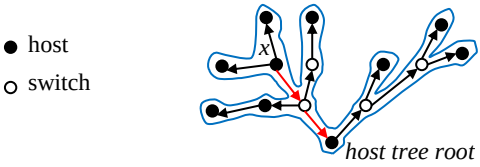


Рис. 6. Для дерева, ориентированного от корня, смена ориентации рёбер при смене корня.
Fig. 6. Changing the edge orientation when the root is changed for a root-oriented tree.

На этапе ***C*** происходит возврат в инициатор, сообщение двигается по циклу хостов, пока не окажется в инициаторе. Инициатор посылает своему внешнему соседу положительный ответ на настройку типа ***BackC***. Напомним, что отрицательный ответ посылается в том случае, когда в сети нет целевых хостов, что обнаруживается на этапе ***A***.

7. Инкрементальная настройка сети

Под инкрементальной настройкой сети так же, как в [1], будем понимать повторную и частичную перенастройку сети, необходимую при изменении сети. В [1] рассматривалась инкрементальная настройка как частичное изменение первичной настройки, т.е. без класса услуг. Основные идеи этой инкрементальной настройки применимы и после введения классов услуг. Нужно только понимать, что изменения сети могут касаться не только цикла хостов и леса деревьев коммутаторов, но также цикла целевых хостов и леса деревьев нецелевых хостов для того или иного класса услуг. В то же время, правила коммутации без класса услуг и для данного класса услуг, а также для разных классов услуг не пересекаются, так как определяются параметрами коммутации: пустым параметром — правила вида (: *a*, *s*, *b*), или классом услуг *class* — правила вида (*class*: *a*, *s*, *b*). Поэтому в данной статье мы не рассматриваем инкрементальную настройку без класса услуг (она такая же, как в [1]) и ограничиваемся рассмотрением инкрементальной настройки для *одного* класса услуг. При некоторых изменениях сети такую инкрементальную настройку нужно выполнить для каждого класса услуг, который затрагивается этим изменением.

Изменение сети касается либо 1) распределения реализаций услуг по хостам сети, либо 2) топологии сети (изменение графа сети), либо 3) распределения услуг по классам. Каждое такое изменение можно представить как последовательность элементарных изменений. Поэтому мы будем рассматривать только элементарные изменения и соответствующие им инкрементальные настройки сети.

1) При изменении распределения реализаций услуг по хостам сети элементарными изменениями являются: 1.1) добавление реализации одной услуги в один хост, 1.2) удаление реализации одной услуги из одного хоста.
2) При изменении топологии сети элементарными изменениями являются: 2.1) изменение упорядочивания графа, 2.2) добавление одного ребра, 2.3) удаление одного ребра, не нарушающее связность графа, 2.4) добавление одного коммутатора

или нецелевого хоста и одного ребра, соединяющего его со «старой» вершиной графа, 2.5) добавление одного целевого хоста и одного ребра, соединяющего его со «старой» вершиной графа, 2.6) удаление одного терминального коммутатора или нецелевого хоста и инцидентного ему ребра, 2.7) удаление одного терминального целевого хоста и инцидентного ему ребра.

3) При изменении распределения услуг по классам элементарными изменениями являются: 3.1) удаление одной услуги из класса, 3.2) добавление одной услуги в класс, 3.3) удаление пустого класса, 3.4) добавление пустого класса.

Примером более сложного изменения сети является удаление целевого хоста и всех инцидентных ему рёбер. Ему соответствует последовательность удалений одного ребра (2.3) для всех инцидентных хосту рёбер, кроме одного, а затем удаление одного терминального целевого хоста и инцидентного ему ребра (2.7). Другой пример — удаление класса услуг, которому соответствует последовательность удалений услуг из класса (3.1), пока он не станет пустым, а затем удаление пустого класса (3.3).

В отличие от предыдущих разделов здесь мы будет давать только идеи алгоритмов настройки, по которым легко можно разработать сами алгоритмы инкрементальной настройки, мы не приводим эти алгоритмы в Приложении. В некоторых случаях перенастройка сети из-за какого-то изменения может быть выполнена эффективнее, если её выполнять «сразу», а не как последовательность перенастроек сети по элементарным изменениям. Но это уже вопрос оптимизации (и усложнения) алгоритмов инкрементальной настройки. Кроме того, мы рассматриваем только такую инкрементальную настройку, которая необходима для поддержания функциональности сети, т.е. возможности передавать сообщения от любого хоста-отправителя любому известному хосту или с запросом любой услуги. Если перенастройка полезна для оптимизации сети (например, уменьшения длины цикла хостов), то мы только отмечаем этот факт, но не предлагаем алгоритмы такой перенастройки.

В ряде случаев во время инкрементальной настройки нужно знать, лежит ли вершина на цикле целевых хостов или нет. У нас нет соответствующей отметки (булевской переменной) в вершинах графа, но без неё можно обойтись. Вершина лежит на цикле целевых хостов тогда и только тогда, когда она является целевым хостом или это коммутатор или нецелевой хост, и в нём либо определено больше одного правила коммутации (при использовании правила умолчания, рис. 5 внизу), либо в его правилах есть хотя бы два разных преемника (без использования правила умолчания, рис. 5 вверху). Напомним, что для цикла целевых хостов на рис. 5 чёрный кружок нужно понимать как целевой хост, белый кружок как коммутатор или нецелевой хост, а серый кружок как коммутатор или (любой) хост.

Инкрементальная настройка сети, как правило, не нарушает функциональности сети. Если во время настройки имеется сообщение указанному хосту (***MessageToHost*** или ***RootMessageToHost***), оно будет доставлено получателю, если в сообщении верно указан его идентификатор и получатель не удалён из сети. Иначе сообщение с отрицательным ответом (***MessageToHost*** или ***RootMessageToHost***), будет доставлено отправителю исходного сообщения, если в сообщении верно указан идентификатор отправителя, и отправитель не удалён из сети. Если во время настройки имеется сообщение с запросом услуги (***Message***, ***RootMessage*** или ***WaitingMessage***), оно будет доставлено нужному хосту, если в сообщении верно указаны имя услуги и имя её класса, и в сети есть хосты, в которых реализована эта услуга. Иначе сообщение с отрицательным ответом (***MessageToHost*** или ***RootMessageToHost***), будет доставлено отправителю исходного сообщения, если в сообщении верно указан идентификатор отправителя и отправитель не удалён из сети. Исключением является случай, когда меняется корень дерева (целевых) хостов: «старый» корень удаляется, и корнем становится другой хост. В этом случае сообщение может не дойти до адресата, даже если такой адресат есть в сети.

7.1 Изменение распределения реализаций услуг по хостам

7.1.1 Добавление реализации одной услуги в один хост

При добавлении реализации услуги с именем *service* в некоторый хост a нужно выполнить в хосте a *Services* := *Services* $\cup$ {*service*}. Если в хосте a была реализована другая услуга того же класса, никаких действий не требуется. В противном случае хост a становится новым целевым хостом для этой услуги, и его нужно добавить в цикл целевых хостов. Для этого на пути по дереву нецелевых хостов от хоста a до цикла целевых хостов нужно поменять правила коммутации в вершинах пути, кроме вершины a, у которой не меняется родитель, и поэтому с учётом правила умолчания в ней изменения не нужны (рис. 7).

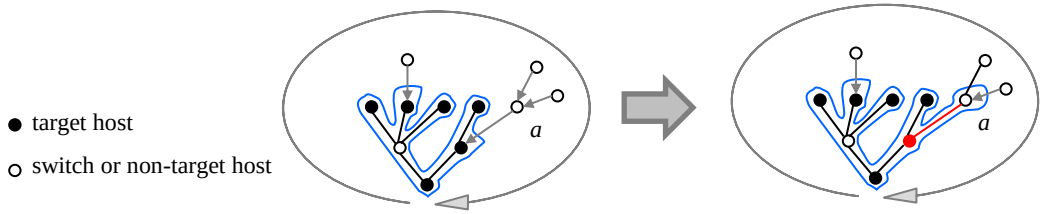


Рис. 7. Нецелевой хост становится целевым хостом.
Fig. 7. A non-target host becomes a target host.

7.1.2 Удаление реализации одной услуги из одного хоста

При удалении реализации услуги с именем *service* из некоторый хоста a нужно выполнить в хосте a *Services* := *Services* \ {*service*}. Если в хосте a осталась реализация другой услуги того же класса, никаких действий не требуется. В противном случае хост a перестаёт быть целевым хостом для этой услуги, и его можно было бы удалить из цикла целевых хостов. Однако это нужно только в целях оптимизации (уменьшения длины цикла целевых хостов), поскольку наличие нецелевых хостов в цикле целевых хостов не влияет на функциональность сети. Относительно просто такую оптимизацию можно выполнить, когда хост a является листовой вершиной дерева целевых хостов, т.е. ему инцидентно в дереве единственное ребро $\{a, b\}$: меняются правила коммутации в вершине b (рис. 8).

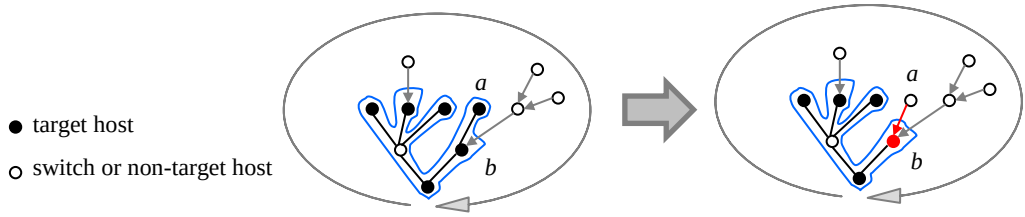


Рис. 8. Листовой целевой хост становится нецелевым хостом.
Fig. 8. A leaf target host becomes a non-target host.

7.2 Изменение топологии сети

7.2.1 Изменение упорядочивания графа

Другое упорядочивание графа, т.е. определение других линейных порядков соседей каждой вершины, приводит к построению другого остовного дерева при настройке сети и, как следствие, других дерева и цикла целевых хостов и леса деревьев нецелевых хостов. Для того чтобы задать новое упорядочивание графа, нужно в каждой вершине, в которой изменился линейный порядок соседей, установить новый список идентификаторов соседних вершин

(отличающийся от старого порядком вершин). Однако для сохранения функциональности сети перенастройка не требуется. В то же время полная перенастройка сети могла бы быть полезна в целях оптимизации: уменьшения длины циклов. Тогда нужно выполнить первичную настройку и далее настройку для каждого класса услуг.

7.2.2 Добавление одного ребра

Для сохранения функциональности сети перенастройка не требуется. В то же время перенастройка сети могла бы быть полезна в целях оптимизации: уменьшения длины цикла целевых хостов. Для подготовки к следующей настройке при добавлении ребра $\{a, b\}$ в список соседей вершины a добавляется идентификатор соседа b, а в список соседей вершины b добавляется идентификатор соседа a.

7.2.3 Удаление одного ребра, не нарушающее связность графа

Для подготовки к следующей настройке при удалении ребра $\{a, b\}$ из списка соседей вершины a удаляется идентификатор соседа b, а из списка соседей вершины b удаляется идентификатор соседа a.

Если удаляемое ребро является хордой остовного дерева, перенастройка не требуется (даже для оптимизации).

Если удаляемое ребро лежит на дереве нецелевых хостов, его удаление нарушает связность этого дерева (рис. 9). Если это ребро $\{a, b\}$, где вершина a является родителем для вершины b, то одна из этих компонент связности является поддеревом остовного дерева с корнем в вершине b, а другая компонента содержит цикл хостов. Поскольку граф остаётся связным, должно быть ребро $\{c, d\}$, соединяющее вершину c из первой компоненты с вершиной d из второй компоненты. Требуется изменить правила коммутации в вершинах пути по (неориентированному) дереву нецелевых хостов из вершины b в вершину c, а также в вершинах a и d. Однако с учётом правила умолчания правила в вершинах a и d можно не менять, так как в них не изменились родители по соответствующим деревьям (дерева хостов для вершины a и дерева коммутаторов для вершины d).

Если удаляемое ребро лежит на дереве целевых хостов, его удаление нарушает связность этого дерева (рис. 10). Если это ребро $\{a, b\}$, где вершина a является родителем для вершины b, то одна из этих компонент связности является поддеревом остовного дерева с корнем в вершине b, а другая компонента содержит корень дерева целевых хостов. Поскольку граф остаётся связным, должно быть ребро $\{c, d\}$, соединяющее вершину c из первой компоненты с вершиной d из второй компоненты. Требуется изменить правила коммутации в вершинах пути по (неориентированному) остовному дереву из вершины b в вершину c и в вершинах пути по (неориентированному) остовному дереву из вершины d до цикла хостов, а также в вершине a.

7.2.4 Добавление одного коммутатора или нецелевого хоста и одного ребра, соединяющего его со «старой» вершиной графа

Для подготовки к следующей настройке при добавлении коммутатора или нецелевого хоста a и ребра $\{a, b\}$ создаётся список (b) соседей вершины a и в список соседей вершины b добавляется идентификатор соседа a. Если добавляется коммутатор, для сохранения функциональности сети перенастройка не требуется. Если добавляется нецелевой хост, его нужно включить в дерево нецелевых хостов, а также сообщить ему имена услуг класса (для отображения услуг в класс). В любом случае при дальнейших изменениях топологии сети во время соответствующей перенастройки сети нужно учитывать добавляемые вершину и ребро. Поскольку при добавлении коммутатора или нецелевого хоста a только с одним инцидентным ему ребром $\{a, b\}$, вершина a будет терминальной, она может входить только в дерево нецелевых хостов. Правила коммутации устанавливаются в добавляемой вершине a

и меняются в другом конце b добавляемого ребра (рис. 11). Однако с учётом правила умолчания достаточно только в вершине a установить правило коммутации {: b, a, b}.

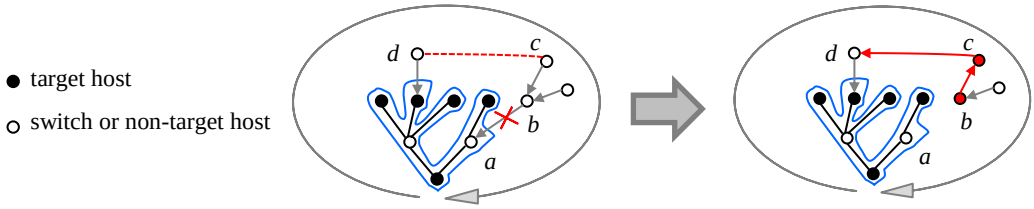


Рис. 9. Удаление ребра на дереве нецелевых хостов.
Fig. 9. Deleting an edge of the non-target host tree.

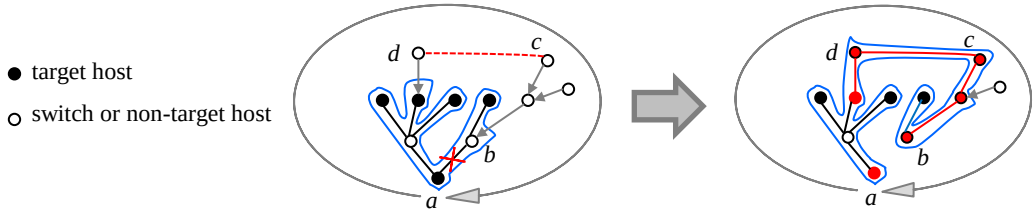


Рис. 10. Удаление ребра на дереве целевых хостов.
Fig. 10. Deleting an edge of the target host tree.

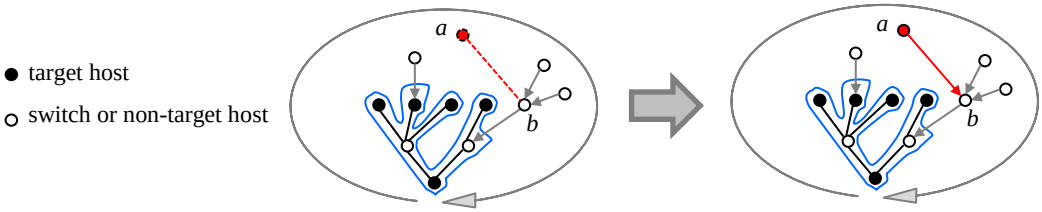


Рис. 11. Добавление одного коммутатора или нецелевого хоста и одного ребра, соединяющего его со «старой» вершиной графа.
Fig. 11. Adding a switch or a non-target host and an edge for its connection with the "old" node of the graph.

7.2.5 Добавление одного целевого хоста и одного ребра, соединяющего его со «старой» вершиной графа

Когда добавляется целевой хост a и ребро {a, b}, соединяющее его со «старой» вершиной b, нужно создать список (b) соседей хоста a, в список соседей вершины b добавить идентификатор соседа a и добавить хост a в цикл целевых хостов. Кроме того, нужно сообщить хосту a имена услуг класса (для отображения услуг в класс). Правила коммутации устанавливаются в добавляемом хосте a, и меняются в вершине b; кроме того, если вершина b не лежит на цикле целевых хостов, то она является некорневой вершиной некоторого дерева нецелевых хостов, и нужно поменять правила коммутации на всём пути от вершины b до корня этого дерева (рис. 12).

7.2.6 Удаление одного терминального коммутатора или нецелевого хоста и инцидентного ему ребра

Связность графа не нарушается, перенастройка не требуется. Для подготовки к следующей настройке при удалении терминального коммутатора или нецелевого хоста a и ребра {a, b} из списка соседей вершины b удаляется идентификатор соседа a.

7.2.7 Удаление одного терминального целевого хоста и инцидентного ему ребра

При удалении терминального целевого хоста a и (единственного) инцидентного ему ребра $\{a, b\}$ связность графа не нарушается. Для подготовки к следующей настройке из списка соседей вершины b удаляется идентификатор соседа a. Удаляемый хост нужно удалить из цикла целевых хостов. Поскольку хост терминальный, он является корнем или листовой вершиной дерева целевых хостов. С учётом правила умолчания достаточно изменить правила коммутации в вершине b (рис. 13). Кроме того, если удаляется корень дерева целевых хостов, нужно отметить в качестве корня другой (всё равно какой) целевой хост.

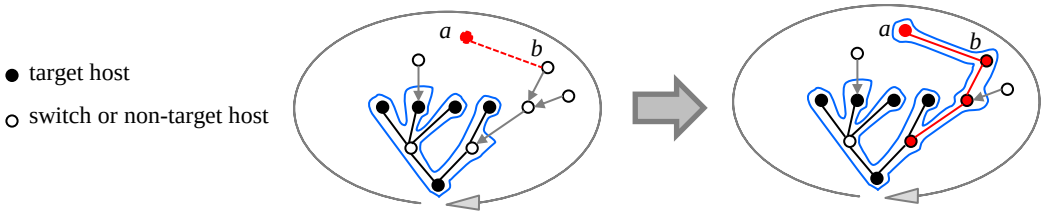


Рис. 12. Добавление одного целевого хоста и одного ребра, соединяющего его со «старой» вершиной графа.
Fig. 12. Adding a target host and an edge connecting that connects this host with the "old" graph node.

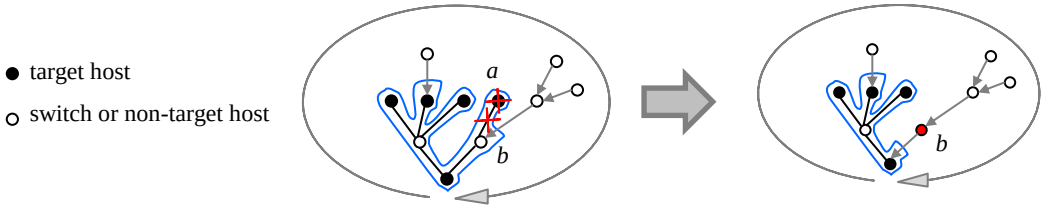


Рис. 13. Удаление одного терминального целевого хоста и инцидентного ему ребра.
Fig. 13. Deleting a terminal target host and its incident edge.

7.3 Изменение распределения услуг по классам

7.3.1 Удаление одной услуги из класса

При удалении услуги из класса нужно сообщить об этом всем хостам, чтобы скорректировать отображение услуг в классы: удалить отображение этой услуги в её класс. Для сохранения функциональности сети перенастройка сети не требуется. Однако в результате такого удаления некоторые хосты, которые были целевыми для данного класса, могут стать нецелевыми (если в них была реализована единственная услуга, удаляемая из класса). Поэтому перенастройка может быть полезна в целях оптимизации: уменьшения длины цикла целевых хостов.

7.3.2 Добавление одной услуги в класс

При добавлении услуги в класс нужно сообщить об этом всем хостам, чтобы скорректировать отображение услуг в классы: добавить отображение этой услуги в её класс. Кроме того, в результате такого добавления некоторые хосты, которые не были целевыми для данного класса, могут стать целевыми (в них реализована добавляемая услуга). Каждый такой хост должен быть включён в цикл хостов, что делается аналогично инкрементальной настройке при добавлении реализации одной услуги в один хост (рис. 7).

7.3.3 Удаление или добавление пустого класса

Никаких действий не требуется.

8. Заключение

Можно указать следующие направления исследований в рамках модели распределённой сети, предложенной в [1] и в данной статье:

1) Мы не исследовали детально (инкрементальную) настройку сети, если она не нужна для сохранения функциональности сети, но полезна для повышения эффективности работы сети, а только указывали на возможную оптимизирующую настройку. Было бы интересно исследовать влияние на эффективность работы сети таких факторов, как распределение услуг по классам и распределение реализаций услуг по хостам в зависимости от топологии сети.
2) Для улучшения эффективности (скорости передачи сообщений) и надёжности сети полезно учитывать соответствующие нефункциональные параметры. Например, можно рассматривать взвешенные графы, в которых рёбрам/вершинам приписаны веса, моделирующие время прохождения сообщения через ребро/вершину (эффективность) или вероятность сбоя при прохождении сообщения через ребро/вершину (надёжность). Соответственно, при построении путей эти параметры желательно учитывать. Для предлагаемых алгоритмов это означает выбор наилучшего дерева хостов, т.е. наилучшего упорядочивания графа и наилучшего корня дерева хостов, выбор наилучшего дерева целевых хостов, т.е. наилучшего корня дерева целевых хостов для данного класса услуг при заданном упорядочивании графа, а также учёт указанных параметров в тех инкрементальных настройках, которые рассмотрены в [1] и этой статье. Также интересно исследовать возможность самонастройки сети: по всей видимости, какие-то решения проблем оптимизации позволяют использовать самонастройку сети, а какие-то нет, требуя предварительного глобального анализа сети (не в процессе самонастройки).

Список литературы / References

[1]. И. Б. Бурдонов, Н. В. Евтушенко, А. С. Косачев, В. Н. Пономаренко. Модель распределённой сети, в которой хосты могут выполнять функцию коммутации сообщений. Труды института системного программирования. 2025, т. 37. № 4, с. 159-172.

[2]. Sezer. S, Scott-Hayward. S, Chouhan P.K., Fraser B., Lake D., Finnegan J., Viljoen N., Miller M. and Rao N. Are we ready for sdn? Implementation challenges for software-defined networks IEEE Communications Magazine, 2013, 51 (7), pp. 36-43.

[3]. Mohammed, A. H., Khaleefah, R. M., k. Hussein, M., and Amjad Abdulateef, I. A review software defined networking for internet of things. In 2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), 2020, pp. 1–8.

[4]. OpenNetworkingFoundation (2012). Software-defined networking: The new norm for networks. ONF White Paper. 2012.

[5]. Burdonov, I.; Kossachev, A.; Yevtushenko, N.; López, J.; Kushik, N. and Zeghlache, D. (2021). Preventive Model-based Verification and Repairing for SDN Requests. In Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE, ISBN 978-989-758-508-1 ISSN 2184-4895, pages 421-428. DOI: 10.5220/0010494504210428.

[6]. Igor Burdonov, Nina Yevtushenko and Alexander Kossatchev. Implementing a virtual network on the SDN data plane. Proceedings 2020 IEEE East-West Design & Test Symposium (EWDTS). 2020, pp. 279-283.

[7]. Бурдонов И.Б., Евтушенко Н.В., Косачев А.С. Реализация распределенных и параллельных вычислений в сети SDN. Труды института системного программирования. 2022, т. 34. № 3, с. 159-172.

ПРИЛОЖЕНИЕ

Типы глобальных переменных и параметров процедур записаны ***строчными буквами полужирным курсивом***.

Имена процедур ***Начинаются с прописной буквы и записаны полужирным курсивом***.

Имена глобальных переменных *Начинаются с прописной буквы и записаны не полужирным курсивом*.
Имена параметров процедур и локальных переменных *начинаются со строчной буквы и записаны не полужирным курсивом*.

П1. Типы глобальных переменных и параметров процедур в хостах и коммутаторах

bool — булевский тип,
vertex — идентификатор вершины графа,
service — имя услуги,
class — имя класса услуг,
set(*type*) — множество элементов типа *type*,
list(*type*) — список элементов типа *type*,
(*type*$_1$, *type*$_2$) — пара (элемент типа *type*$_1$, элемент типа *type*$_2$),
type$_1$→*type*$_2$ — отображение элемента типа *type*$_1$ в элемент типа *type*$_2$, эквивалентно ***set***((*type*$_1$, *type*$_2$)) с ограничением: любое значение встречается в левых частях пар не более одного раза.

П2. Глобальные переменные в вершине — хосте или коммутаторе

vertex *Self* = ...; /* инициализировано, не меняется: собственный идентификатор вершины */
list(***vertex***, ***vertex***) *Rules*; /* список правил коммутации как список пар (предшественник, преемник) */
/* без класса услуг (для цикла хостов и леса деревьев коммутаторов */
class → ***list***(***vertex***, ***vertex***) *ClassToRules*; /* список правил коммутации для данного класса услуг */
bool *Host* = ...; /* инициализировано, не меняется: признак того, что вершина является хостом */

П3. Глобальные переменные в хосте

set(***service***) *Services* = ...; /* инициализировано: множество имён услуг, реализуемых хостом */
class → ***vertex*** *ClassToExternal*; /* идентификатор внешнего соседа инициатора */
/* для данного класса услуг */
class → ***Bool*** *ClassToInitiator*; /* признак инициатора для данного класса услуг */
class → ***Bool*** *ClassToRoot*; /* признак корня дерева целевых хостов для данного класса услуг */
service → ***class*** *ServiceToClass*; /* отображение имени услуги в имя её класса */

П4. Использование глобальных переменных

Глобальные переменные, используемые на этапе самонастройки сети для данного класса услуг: *Rules*, *ClassToRules*, *Host*, *Services*, *ClassToExternal*, *ClassToInitiator*, *ClassToRoot*, *ServiceToClass* (все, кроме *Self*).
Глобальные переменные, используемые при передаче сообщений запроса услуг по настроенной сети: *Self*, *Rules*, *ClassToRules*, *Host*, *Services*, *ClassToRoot* (все, кроме *ClassToExternal*, *ClassToInitiator*, *ServiceToClass*); при генерации сообщений запроса услуг: *ClassToRules*, *ClassToRoot*, *ServiceToClass*.

П5. Правила коммутации

Правила коммутации *rules* = *Rules* без класса услуг или *rules* = *ClassToRules*(*class*) для класса услуг *class* представлены как список длиной *n* пар (идентификатор соседа-предшественника, идентификатор соседа-преемника): (a_0, a_1), (a_1, a_2), ..., (a_{n-1}, a_n). Мы будем использовать обычную нотацию:
rules[*i*..*j*] = (a_{i-1}, a_i), ..., (a_{j-1}, a_j), если $i = 1..n$, $j = 1..n$ и $i \le j$; иначе *rules*[*i*..*j*] = ();
rules[*i*][1] = a_{i-1} для $i = 1..n$;
rules[*i*][2] = a_i для $i = 1..n$.
Правило умолчания: при получении вершиной сообщения от вершины *b*, отличной от вершин a_0, ..., a_{n-1}, оно пересылается родителю a_0 (предшественнику из первого правила) по подразумеваемому правилу (*b*, a_0).

П6. Вспомогательные процедуры

vertex Successor(***list***(***vertex***, ***vertex***) *rules*, ***vertex*** *x*) { /* эта процедура такая же, как в [1] */
    /* по предшественнику *x* вычисление преемника *b* в списке правил *rules* = (a_0, a_1), ..., (a_{n-1}, a_n) */
    n = |*rules*|; *i* : = 1;
    while *i* ≤ *n* & *rules*[*i*][1] ≠ *x* ***do*** { *i* := *i* + 1; }
    if *i* ≤ *n* { ***return*** *rules*[*i*][2]; }
    else return *rules*[1][1]; } /* если для каждого *b* в *rules* нет правила (*x*, *b*), возвращаем a_0 */

list(***vertex***, ***vertex***) ***RulesReordering***(***vertex*** *x*) /* изменение начала циклического списка правил */
    n = |*Rules*|; *i* : = 1; /* первым правилом становится правило вида (*x*, *b*) */
    while *i* ≤ *n* & *Rules*[*i*][1] ≠ *x* ***do*** { *i* := *i* + 1; }
    return *rules*[*i*][*n*]^*rules*[1][*i* - 1];

П7. Процедуры обработки сообщений

Сигнатура процедуры обработки сообщения имеет вид ***Type***(***vertex*** *x*, *параметры сообщения*), а оператор посылки сообщения имеет вид ***SEND***(***Type***(*параметры сообщения*), *y*), где ***Type*** тип сообщения, x идентификатор соседа-предшественника, от которого принято сообщение, y идентификатор соседа-преемника, которому посылается сообщение.

Предполагается, что выполнена первичная настройка: построены цикл хостов и лес деревьев коммутаторов, в каждой вершине инициализированы переменные *Self*, *Rules*, *Host*, *Root*, *Services*. Алгоритмы первичной настройки приведены в [9], там же приведены процедуры обработки сообщений указанному хосту ***MessageToHost*** и ***RootMessageToHost*** и процедура генерации таких сообщений ***SendMessageToHost***. Процедуры запроса услуг ***Message***, ***RootMessage*** и ***WaitingMessage***, а также процедура ***SendMessage*** генерации сообщения запроса услуг, приведённые ниже, отличаются от тех, что даны в [1].

П7.1 Самонастройка сети для данного класса услуг

vertex *x* — идентификатор соседа, от которого получено сообщение,
class *class* — имя класса услуг,
set(***service***) *serviceSet* — множество имён услуг этого класса.

П7.1.1 Этап A: поиск целевого хоста

ClassStart(***vertex*** *x*, ***class*** *class*, ***set***(***service***) *serviceSet*) {
    /* старт настройки, сообщение пришло в хост-инициатор от его внешнего соседа *x* */
    for ∀ *service* ∈ *serviceSet* ***do*** { /* отображение услуг в их класс */
        ServiceToClass(*service*) : = *class*; }
    if *Rules* = () { /* если инициатор изолированная вершина, то */
        ClassToRules(*class*) := (); /* правил нет */
        if *Services* ∩ *serviceSet* ≠ ∅ { /* если инициатор целевой хост, то */
            ClassToRoot(*class*) := ***true***; /* инициатор становится корнем дерева целевых хостов */
            SEND(***BackC***(*class*, *serviceSet*), *x*); } /* положительный ответ на настройку */
        else { /* если инициатор не целевой хост, то */
            ClassToRoot(*class*) := ***false***; /* инициатор не является корнем дерева целевых хостов */
            SEND(***CancelC***(*class*, *serviceSet*), *x*); } } /* отрицательный ответ на настройку */
    else { /* если инициатор не изолированная вершина, то */
        ClassToInitiator(*class*) := ***true***; /* хост становится инициатором */
        ClassToExternal(*class*) := *x*; /* запоминаем идентификатор *x* внешнего соседа инициатора */
        if *Services* ∩ *serviceSet* ≠ ∅ { /* если инициатор целевой хост, то */
            ClassToRules(*class*) := *Rules*; /* сначала цикл целевых хостов совпадает с циклом хостов */
            ClassToRoot(*class*) := ***true***; /* инициатор становится корнем дерева целевых хостов */

```
        /* прямое сообщение этапа B соседу a1 */
        SEND(ForwardB(class, serviceSet), Rules[1][2]); } }
    else { /* если инициатор не целевой хост, то */
        ClassToRules(class) := (); /* сначала цикла целевых хостов нет */
        ClassToRoot(class) := false; /* инициатор не является корнем дерева целевых хостов */
        /* поиск целевого хоста, начиная с соседа a1 */
        SEND(SearchA(class, serviceSet), Rules[1][2]); } } }

SearchA(vertex x, class class, set(service) serviceSet) { /* этап A, поиск целевого хоста */
    y := Successor(Rules, x); /* сосед-преемник y соседа-предшественника x в цикле хостов */
    if x = Rules[1][1] { /* если пришли в вершину от её родителя a0 = x */
        if Host = false ∨ ClassToInitiator(class) = false {
            /* если вершина не инициатор, то первый раз на этапе A пришли в вершину, тогда */
            if Host = true { /* если вершина хост, то */
                for ∀ service ∈ serviceSet do { /* отображение услуг в их класс */
                    ServiceToClass(service) : = class; }
            if Host = true & Services ∩ serviceSet ≠ ∅ { /* если вершина целевой хост, то */
                ClassToRules(class) := Rules; /* сначала цикл целевых хостов равен циклу хостов */
                ClassToRoot(class) := true; /* вершина становится корнем дерева целевых хостов */
                /* прямое сообщение этапа B вершине y = a1 */
                SEND(ForwardB(class, serviceSet), Rules[1][2]); }
            else { /* если вершина не целевой хост, то */
                ClassToRules(class) := (); /* сначала цикла целевых хостов нет */
                if Host = true { /* если вершина хост, то */
                    ClassToRoot(class) := false; /* очистка признака корня целевых хостов */
                SEND(SearchA(class, serviceSet), y); } } /* идём дальше по циклу хостов */
        else { /* если пришли в инициатор, то цикл хостов пройден и целевых хостов нет */
            ClassToInitiator(class) := false; /* конец настройки, очистка для следующей настройки */
            SEND(CancelC(class, serviceSet), External(class)); } /* отрицательный ответ на настройку */
    else { /* если цикл хостов не пройден, то */
        SEND(SearchA(class, serviceSet), y); } } /* идём дальше по циклу хостов */
```

П7.1.2 Этап B: построение цикла целевых хостов и деревьев коммутаторов

```
ForwardB(vertex x, class class, set(service) serviceSet) { /* этап B, прямое сообщение */
    /* первое сообщение, получаемое вершиной на этапе B, новым родителем становится вершина x */
    ClassToRules(class) := RulesReordering(x); /* меняем начало циклического списка правил (a0 = x) */
    if Host = true { /* если вершина хост, то запоминаем отображение услуг в их класс */
        for ∀ service ∈ serviceSet do { ServiceToClass(service) : = class; }
        ClassToRoot(class) := false; } /* очистка признака корня целевых хостов */
    if |ClassToRules(class)| = 1 { /* если это листовая вершина дерева хостов, то */
        if Host = true & Services ∩ serviceSet ≠ ∅ { /* если это целевой хост, то */
            SEND(BackB(class, serviceSet), x); } /* положительный ответ родителю a0 = x */
        else { /* если это не целевой хост, то */
            SEND(CancelB(class, serviceSet), x); } } /* отрицательный ответ родителю a0 = x */
    else { /* если это не листовая вершина дерева хостов, то */
        /* прямое сообщение соседу a1 */
        SEND(ForwardB(class, serviceSet), ClassToRules(class)[1][2]); } }
```

```
CancelB(vertex x, class class, set(service) serviceSet) { /* этап B, отрицательный ответ */
    y := Successor(ClassToRules(class), x); /* преемник y предшественника x в цикле правил */
    /* Корректировка правил коммутации для данного класса услуг */
    n := |ClassToRules(class)|; i := 1; /* определяем индекс правила, в котором вершина x преемник */
    while i ≤ n & ClassToRules(class)[i][2] ≠ x do { i := i + 1; }
    if i < n { /* x = a_i, i < n, текущая вершина не корень */
        /* меняем правило (a_{i-1}, a_i) → (a_{i-1}, a_{i+1}) */
        ClassToRules(class)[i][2] := ClassToRules(class)[i + 1][2]; {
        /* удаляем правило (a_i, a_{i+1}) */
        ClassToRules(class) := ClassToRules(class)[1..i]^ClassToRules(class)[i + 2..n]; }
    else { /* если x = a_n = a_0, то текущая вершина корень, тогда */
        ClassToRules(class)[1][1] := ClassToRules(class)[n][1]; /* меняем правило (a_0, a_1) → (a_{n-1}, a_1) */
        ClassToRules(class) := ClassToRules(class)[1..n - 1]; } /* удаляем правило (a_{n-1}, a_0) */
    CancelBackB(y, class, serviceSet); } /* окончание в процедуре CancelBackB */
BackB(vertex x, class class, set(service) serviceSet) { /* этап B, положительный ответ */
    y := Successor(ClassToRules(class), x); /* преемник y предшественника x в цикле правил */
    CancelBackB(y, class, serviceSet); } /* окончание в процедуре CancelBackB */
CancelBackB(vertex y, class class, set(service) serviceSet) { /* окончание процедур CancelB и BackB */
    if y = ClassToRules(class)[1][2] { /* если y = a_1, то прошли цикл хостов и пришли в корень */
        if ClassToInitiator(class) = true { /* если инициатор, то */
            ClassToInitiator(class) := false; /* конец настройки, очистка для следующей настройки */
            /* положительный ответ внешнему соседу */
            SEND(BackC(class, serviceSet), ClassToExternal(class)); }
        else { /* если не инициатор, то положительный ответ посылается инициатору */
            SEND(BackC(class, serviceSet), Rules[1][2]); } } /* по циклу хостов соседу a_1 */
    else { /* если не прошли цикл хостов, то */
        if y = ClassToRules(class)[1][1] {
            /* если y = a_0 (родитель), то прошли поддерево с корнем в этой вершине, тогда */
            if Host = true & Services ∩ serviceSet ≠ ∅ ∨ |ClassToRules(class)| > 1 {
            /* если в поддереве с корнем в данной вершине есть целевой хост то, */
                SEND(BackB(class, serviceSet), y); } /* положительный ответ родителю y = a_0 */
            else { /* если в поддереве с корнем в данной вершине нет целевого хоста, то */
                SEND(CancelB(class, serviceSet), y); } } /* отрицательный ответ родителю y = a_0 */
        else { /* если y ≠ a_0 (не родитель), то не прошли поддерево с корнем в этой вершине */
            /* прямое сообщение этапа B дальше по циклу целевых хостов */
            SEND(ForwardB(class, serviceSet), y); } } }
```

П7.1.3 Этап C: положительный ответ на настройку сети

Отрицательный ответ ***CancelC*** посылается во внешнюю вершину в процедурах ***ClassStart*** и ***SearchA***.

```
BackC(vertex x, class class, set(service) serviceSet) { /* этап C, положительный ответ на настройку */
    y := Successor(Rules, x); /* сосед-преемник y соседа-предшественника x в цикле хостов */
    if Host = true & ClassToInitiator(class) = true { /* если вершина инициатор, то */
        ClassToInitiator(class) := false; /* конец настройки, очистка для следующей настройки */
        /* положительный ответ внешнему соседу */
        SEND(BackC(class, serviceSet), ClassToExternal(class)); }
    else { /* если вершина не инициатор, то */
        SEND(BackC(class, serviceSet), y); } } /* положительный ответ по циклу хостов инициатору */
```

П7.2 Передача сообщений запроса услуг по настроенной сети

Обработка сообщений запроса услуги отличаются от обработки одноимённых сообщений в [1] только добавлением в число параметров параметра ***class*** *class*, а также именами переменных: *ClassToRules*(*class*) вместо *Rules* и *ClassToRoot*(*class*) вместо *Root*.

vertex *x* — идентификатор соседа, от которого получено сообщение,

vertex *sender* — идентификатор хоста-отправителя, применяется для посылки ответа отправителю,

class *class* — имя класса запрашиваемой услуги,

service *service* — имя запрашиваемой услуги,

parameters — параметры сообщения, прозрачные для коммутации сообщений.

```
Message(vertex x, vertex sender, class class, service service, parameters) {
    /* сообщение запроса услуги, не прошедшее через дугу от родителя корня в корень *
    y := Successor(ClassToRules(class), x); /* сосед-преемник y соседа-предшественника x */
    if Host ≠ true ∨ service ∉ Services { /* если вершина не реализует запрашиваемую услугу, то */
        if x ≠ ClassToRules(class)[1][1] ∨ Host ≠ true ∨ Root = false {
            /* если пришли не по дуге от родителя корня в корень, то */
            /* посылаем сообщение дальше по циклу целевых хостов */
            SEND(Message(sender, class, service, parameters), y); }
        else { /* если пришли по дуге от родителя корня в корень, то */
            /* посылаем сообщение дальше по циклу целевых хостов */
            SEND(RootMessage(sender, class, service, parameters), y); } }
    else { /* если вершина хост, реализующий запрашиваемую услугу, то */
        if HostReadyToExecuteService(service, parameters) = false { /* если хост «занят», то */
            /* посылаем сообщение дальше по циклу целевых хостов */
            SEND(WaitingMessage(sender, class, service, parameters), y); }
        else { /* если хост «свободен», то */
            service(sender, parameters); } } } /* локальный вызов запрашиваемой услуги */

RootMessage(vertex x, vertex sender, class class, service service, parameters) {
    /* сообщение запроса услуги, прошедшее через корень дерева хостов *
    y := Successor(ClassToRules(class), x); /* сосед-преемник y соседа-предшественника x */
    if Host ≠ true ∨ service ∉ Services { /* если вершина не реализует запрашиваемую услугу, то */
        if x ≠ ClassToRules(class)[1][1] ∨ Host ≠ true ∨ Root = false {
            /* если пришли не по дуге от родителя корня в корень, то */
            /* посылаем сообщение дальше по циклу целевых хостов */
            SEND(RootMessage(sender, class, service, parameters), y); }
        else { /* если пришли по дуге от родителя корня в корень, то */
            /* отрицательный ответ отправителю сообщения, не нашедшего получателя */
            SEND(MessageToHost(Self, class, sender, parameters), Rules[1][2]); } }
    else { /* если вершина хост, реализующий запрашиваемую услугу, то */
        if HostReadyToExecuteService(service, parameters) = false { /* если хост «занят», то */
            /* посылаем сообщение дальше по циклу целевых хостов */
            SEND(WaitingMessage(sender, class, service, parameters), y); }
        else { /* если хост «свободен», то */
            service(sender, parameters); } } } /* локальный вызов запрашиваемой услуги */

WaitingMessage(vertex x, vertex sender, class class, service service, parameters) {
    /* сообщение запроса услуги, ожидающее освобождения хоста, который может оказать услугу */
    y := Successor(ClassToRules(class), x); /* сосед-преемник y соседа-предшественника x */
```

```
    if Host ≠ true ∨ service ∉ Services { /* если вершина не реализует запрашиваемую услугу, то */
        if x ≠ ClassToRules(class)[1][1] ∨ Host ≠ true ∨ Root = false {
            /* если пришли не по дуге от родителя корня в корень, то */
            /* посылаем сообщение дальше по циклу целевых хостов */
            SEND(WaitingMessage(sender, class, service, parameters), y); }
        else { /* если пришли по дуге от родителя корня в корень, то */
            /* посылаем сообщение дальше по циклу целевых хостов */
            SEND(Message(sender, class, service, parameters), y); } }
    else { /* если вершина хост, реализующий запрашиваемую услугу, то */
        if HostReadyToExecuteService(service, parameters) = false { /* если хост «занят», то */
            /* посылаем сообщение дальше по циклу целевых хостов */
            SEND(WaitingMessage(sender, class, service, parameters), y); }
        else { /* если хост «свободен», то */
            service(sender, parameters); } } } /* локальный вызов запрашиваемой услуги */
```

П8. Вызов удалённой услуги из хоста

service *service* — имя запрашиваемой услуги,
parameters — параметры сообщения, прозрачные для коммутации сообщений.

```
bool SendMessage(service service, parameters); { /* вызов из хоста удалённой услуги */
    class : = ServiceToClass(service); /* определяем класс услуг */
    if ClassToRules(class) = () { /* если нет правил, то */
        return false; }
    y := ClassToRules(class)[1][2]; /* посылаем следующему соседу a1 */
    if service ∉ Services { /* если в хосте не реализована запрашиваемая услуга, то */
        if ClassToRoot(class) = true { /* если хост корень дерева целевых хостов, то */
            SEND(RootMessage(Self, class, service, parameters), y); } /* по циклу целевых хостов */
        else { /* если хост не корень цикла целевых хостов, то */
            /* посылаем сообщение по пути до цикла целевых хостов и далее по циклу */
        SEND(Message(Self, class, service, parameters), y); }
    else { /* если в хосте реализована запрашиваемая услуга (но хост сейчас занят), то */
            /* посылаем сообщение по циклу целевых хостов */
        SEND(WaitingMessage(Self, class, service, parameters), y); }
    return true; }
```

Информация об авторах / Information about authors

Игорь Борисович БУРДОНОВ – доктор физико-математических наук, главный научный сотрудник ИСП РАН. Научные интересы: формальные спецификации, генерация тестов, технология компиляции, системы реального времени, операционные системы, объектно-ориентированное программирование, сетевые протоколы, процессы разработки программного обеспечения.

Igor Borisovich BURDONOV – Dr. Sci. (Phys.-Math.), a Leading Researcher of ISP RAS. Research interests: formal specifications, test generation, compilation technology, real-time systems, operating systems, object-oriented programming, network protocols, software development processes.

Нина Владимировна ЕВТУШЕНКО, доктор технических наук, профессор, главный научный сотрудник ИСП РАН, до 1991 года работала научным сотрудником в Сибирском физико-

техническом институте. С 1991 г. работала в ТГУ профессором, зав. кафедрой, зав. лабораторией по компьютерным наукам. Её исследовательские интересы включают формальные методы, теорию автоматов, распределённые системы, протоколы и тестирование программного обеспечения.

Nina Vladimirovna YEVTUSHENKO, Dr. Sci. (Tech.), Professor, a Leading Researcher of ISP RAS, worked at the Siberian Scientific Institute of Physics and Technology as a researcher up to 1991. In 1991, she joined Tomsk State University as a professor and then worked as the chair head and the head of Computer Science laboratory. Her research interests include formal methods, automata theory, distributed systems, protocol and software testing.

Александр Сергеевич КОСАЧЕВ – кандидат физико-математических наук, ведущий научный сотрудник ИСП РАН. Научные интересы: формальные спецификации, генерация тестов, технология компиляции, системы реального времени, операционные системы, объектно-ориентированное программирование, сетевые протоколы, процессы разработки программного обеспечения.

Alexander Sergeevitch KOSSATCHEV – Cand. Sci. (Phys.-Math.), a Leading Researcher of ISP RAS. Research interests: formal specifications, test generation, compilation technology, real-time systems, operating systems, object-oriented programming, network protocols, software development processes.

Вера Николаевна ПОНОМАРЕНКО – кандидат физико-математических наук, старший научный сотрудник ИСП РАН. Научные интересы: формальные спецификации, генерация тестов, системы реального времени, операционные системы, объектно-ориентированное программирование, сетевые протоколы, процессы разработки программного обеспечения.

Vera Nikolaevna PONOMARENKO – Cand. Sci. (Phys.-Math.), a Senior Researcher of ISP RAS. Research interests: formal specifications, test generation, real-time systems, operating systems, object-oriented programming, network protocols, software development processes.

DOI: 10.15514/ISPRAS-2025-37(5)-2

Применение кодов в модульной метрике для поиска k-соседей

А.Р. Шарапов, ORCID: 0000-0002-4794-0206 <arsharapov@hse.ru>
В.А. Давыдов, ORCID: 0000-0003-1316-3346 <v.davydov@hse.ru>

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

Аннотация. Рассматривается применение кодов, исправляющих ошибки в модульной метрике, для решения задачи идентификации объекта на множестве Q-ичных векторов размерности D методом k-соседей. В качестве предварительной обработки обучающей выборки используется метод кластеризации, использующий процедуру декодирования всех векторов обучающей выборки выбранным кодом в модульной метрике.

Ключевые слова: метод k-ближайших соседей; метрики; кластеризация; коды в модульной метрике; вектор.

Для цитирования: Шарапов А.Р., Давыдов В.А. Применение кодов в модульной метрике для поиска k-соседей. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 33–42. DOI: 10.15514/ISPRAS–2025–37(5)–2.

Благодарности. Статья подготовлена в ходе проведения исследования в рамках Программы фундаментальных исследований Национального исследовательского университета «Высшая школа экономики» (НИУ ВШЭ).

Application of Codes in Modular Metrics for Searching K-Neighbors

A.R. Sharapov, ORCID: 0000-0002-4794-0206 <arsharapov@hse.ru>
V.A. Davydov, ORCID: 0000-0003-1316-3346 <petrov@ispras.ru>

National Research University Higher School of Economics, 11, Pokrovksy Bulvar, Moscow, 109028, Russia.

Abstract. This paper is devoted to the application of suffix codes in the modular metric for solving clustering and k-nearest neighbors (KNN) problems. The advantages of using the modular metric over the Euclidean metric are considered, especially in high-dimensional spaces. The main emphasis is placed on the development of efficient clustering and k-nearest neighbors algorithms using codes that can correct errors in the modular metric. The proposed approach provides polynomial complexity with respect to the training sample dimension, which makes it promising for machine learning applications with large datasets and high-performance requirements.

Keywords: KNN (k-nearest neighbors) method; metrics; clustering; codes in module metric; vector.

For citation: Sharapov A.R., Davydov V.A. Application of codes in modular metrics for searching k-neighbors. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 33-42 (in Russian). DOI: 10.15514/ISPRAS-2025-37(5)-2.

Acknowledgements. The article was prepared during the research conducted within the framework of the Fundamental Research Program of the National Research University Higher School of Economics (HSE).

1. Введение

Классическая проблема поиска ближайшего соседа формулируется следующим образом: задано подмножество $\boldsymbol{V}$ вещественного векторного пространства размерности D с заданной метрикой ρ и состоящее из N векторов. Также имеется подмножество $\boldsymbol{M}$ векторного пространства $\boldsymbol{V}$ и элемент $\boldsymbol{b} \in \boldsymbol{V}$, требуется найти $\boldsymbol{a} \in \boldsymbol{M}$, ближайший к $\boldsymbol{b}$. Данная проблема и ее решение является центральной проблемой в вычислительной геометрии.

Быстрое вычисление ближайших соседей активно изучается в рамках научных направлений машинного обучения. Наиболее наивная реализация поиска соседей включает в себя грубое вычисление расстояний между всеми парами точек в наборе данных, этот подход имеет сложность $O(DN^2)$. Однако, поскольку количество выборок N растет, подход грубой силы быстро становится невозможным.

Для решения проблемы вычислительной неэффективности подхода грубой силы были изобретены различные древовидные структуры данных. Эффективное кодирование совокупной информации о расстоянии позволяет сократить необходимое количество вычислений расстояния в выборке благодаря использованию древовидных структур данных. Основная идея заключается в том, что если точка A очень далека от точки B, и точка B очень близка к C, то мы знаем, что точки A и C очень далеки, без необходимости явно вычислять их расстояние. Таким образом, вычислительная стоимость поиска ближайших соседей может быть снижена до $O(DN \log N)$ или лучше. Это значительное улучшение по сравнению с грубой силой для больших N.

В литературе также изучается модифицированная задача поиска приблизительного ближайшего соседа. В приблизительном поиске соседа на расстоянии r, структура данных должна сообщать о точке на расстоянии cr от заданной точки $\boldsymbol{b}$ для некоторой константы $c > 1$, но только в случае, если существует точка на расстоянии r от $\boldsymbol{b}$. Будем называть эту проблему (r, c) -ближайшим соседом (nearest neighbor, NN)

В работе [6] представлен LSH алгоритм поиска ближайшего соседа, который использует $O\left(DN^{1+1/c}\right)$ подготовительных операций, объем памяти $O(DN + N^{1+1/c})$ и $O(DN^{1/c})$ операций по поиску ближайшего сосед. Используя уменьшение размерности [7], число операций поиска ближайшего соседа можно дополнительно сократить до $O(D + N^{1/c})$, а сложность предварительной обработки сократить до $O\left(DN + N^{1+1/c}\right)$. Алгоритм LSH успешно использовался в нескольких прикладных сценариях, включая вычислительную биологию [8-9] и ссылки в них или [10: 414].

В работе [1] был представлен новый подход к решению задачи поиска K-ближайших соседей (KNN) [5] для случая модульной метрики, который предлагает использование кодовых конструкций. Предложенный метод свел задачу классификации к процедуре декодирования кодов в модульной метрике, которая была описана в работе [2]. Модульную метрику также называют метрикой Манхэттена или $l1$ метрикой.

Как было отмечено в [1], модульная метрика имеет ряд преимуществ по сравнению с метрикой Евклида для пространств с большой размерностью, что дает дополнительный аргумент для ее использования в задачах классификации на таких пространствах. Метод K-ближайших соседей, как отмечено в [4] является наиболее точным для решения задач классификации по сравнению с альтернативными методами, однако его применение ограничено алгоритмической сложностью, которая у известных вариантов реализации KNN зависит от объема обучающей выборки.

Предположим, что объем обучающей выборки $\boldsymbol{V}$ равен N. Для поиска ближайшего расстояния необходимо перебрать все объекты из обучающей выборки, рассчитать для каждого из них расстояние до тестового объекта $\boldsymbol{f}$ и затем найти минимум. Сложность такого поиска линейная по N и зависит от размерности пространства признаков D.

Предположим, что значение каждого признака имеет определенные границы т.е. минимальное значение и максимальное значение. Разницу между максимальным и минимальными значениями будем называть диапазоном признака. Проведем операцию нормирования каждого признака, то есть приведем минимальное значение к нулю, а максимальное к единице путем вычитания минимального значения из всех значений и деления результата на диапазон. Для перехода к целым числам будем считать, что значения отрезка от нуля до единицы разделены на Z равномерных интервалов. Пусть $\{0, 1, 2, \dots, Z-1\} \equiv Z$ Если $\boldsymbol{f} \in Z^D$, то сложность такого алгоритма поиска $O(ND)$. В типичной задаче машинного обучения количество признаков D может быть порядка 100, а размер выборки может исчисляться десятками и сотнями тысяч объектов. Такая сложность является сдерживающим фактором для реализации метода в приложениях промышленного интернета вещей на устройствах, где нужно малое время реакции системы, энергоэффективность и низкие требования к «железу». Всё это означает, что для решения проблемы KNN возникает необходимость в более быстрых методах поиска ближайших соседей, чем простой перебор.

Переход к кодам в модульной метрике, предложенный в работе [1], дает возможность решения задачи классификации методом KNN с полиномиальной сложностью от размерности обучающей выборки, что открывает новые перспективы для использования KNN в машинном обучении, особенно для больших объемов данных и больших размерностей выборки. Использование кодов в модульной метрике в конкретных приложениях для KNN зависит от того, насколько эффективность теоретического подхода, описанного в работе [1], будет сохранена при практической реализации. Для такой реализации в данной работе предлагается использование суффиксной конструкции кодов в модульной метрике, которая была описана в статье [3]

Сложность решения задачи поиска ближайшего соседа для выборки $\mathbf{V}$ размерности D с заданной метрикой ρ и состоящее из N векторов при больших значениях N может быть существенно снижена при проведении предварительной обработки выборки $\mathbf{V}$**.** В качестве такой обработки может быть использована кластеризация выборки $\mathbf{V}$**,** описание которой приводится в следующем разделе.

2. Задача кластеризации

Задача кластеризации (или обучения без учителя) заключается в следующем. Задано множество $\boldsymbol{V}$ размерности D с заданной метрикой ρ, и состоящее из N векторов. Требуется разбить выборку на непересекающиеся подмножества, называемые кластерами, так, чтобы каждый кластер состоял из объектов, близких по метрике ρ, а объекты разных кластеров существенно отличались. При этом каждому объекту $\boldsymbol{a} = (a_1, a_2, \dots, a_D) \in \boldsymbol{V}$ приписывается метка (номер) кластера $\boldsymbol{y_a} \in \boldsymbol{Y}$. Алгоритм кластеризации — это функция $\boldsymbol{V} \to \boldsymbol{Y}$, которая любому объекту $\boldsymbol{a} = (a_1, a_2, \dots, a_D) \in \boldsymbol{V}$ ставит в соответствие метку кластера $\boldsymbol{y_a} \in \boldsymbol{Y}$. Множество меток Y в некоторых случаях известно заранее, однако чаще ставится задача определить оптимальное число кластеров, с точки зрения того или иного критерия качества кластеризации.

Имеются многочисленные обзоры работ в области кластерного анализ, например, статьи [11-14]. Ляо в 2005 году [16] опубликовал обзор методов кластеризации для данных временных рядов.

Решение задачи кластеризации принципиально неоднозначно, и тому есть несколько причин. Во-первых, не существует однозначно наилучшего критерия качества кластеризации. Известен целый ряд достаточно разумных критериев, а также ряд алгоритмов, не имеющих чётко выраженного критерия, но осуществляющих достаточно разумную кластеризацию «по построению». Все они могут давать разные результаты. Во-вторых, число кластеров, как правило, неизвестно заранее и устанавливается в соответствии с некоторым субъективным

критерием. В-третьих, результат кластеризации существенно зависит от метрики ρ, выбор которой, как правило, также субъективен и определяется экспертом.

Для решения в дальнейшем задачи поиска ближайшего соседа мы будем использовать такие цели кластеризации как:

- упростить дальнейшую обработку данных и принятия решений, работая с каждым кластером по отдельности;
- сократить объём хранимых данных в случае сверхбольшой выборки $\boldsymbol{V}$, оставив по одному наиболее типичному представителю от каждого кластера.

В первом случае число кластеров стараются сделать поменьше. Во втором случае важнее обеспечить высокую степень сходства объектов внутри каждого кластера, а кластеров может быть сколько угодно.

Во всех этих случаях может применяться иерархическая кластеризация, когда крупные кластеры дробятся на более мелкие, те в свою очередь дробятся ещё мельче и так далее. Такие задачи называются задачами таксономии (taxonomy). Результатом таксономии является не простое разбиение множества объектов на кластеры, а древообразная иерархическая структура. Вместо номера кластера объект характеризуется перечислением всех кластеров, которым он принадлежит, от крупного к мелкому. Мы будем рассматривать алгоритмы иерархической кластеризации, позволяющие автоматизировать процесс построения таксономий.

Задачу кластеризации можно ставить как задачу дискретной оптимизации: необходимо так приписать номера кластеров $\boldsymbol{y_a} \in \boldsymbol{Y}$ каждому объекту $\boldsymbol{a} = (a_1, a_2, \dots, a_D) \in \boldsymbol{V}$, чтобы значение выбранного функционала качества приняло наилучшее значение. Существует много разновидностей функционалов качества кластеризации, но нет «самого правильного» функционала.

Если алгоритм кластеризации вычисляет центры кластеров $\mu(\boldsymbol{y_a})$, $\boldsymbol{y_a} \in \boldsymbol{Y}$, то можно определить функционал суммы средних внутрикластерных расстояний:

$$\Phi_0(\mu, V) = \sum_{y \in Y} \frac{1}{|K_y|} \sum_{a|y_a=y} \rho(a, \mu(y_a)) \to min$$

где $K_y = \{a \in V | y_a = y\}$ — кластер с номером y.

Также можно определить функционал суммы межкластерных расстояний:

$$\Phi(\mu) = \sum_{z_1, z_2 \in \mu(Y)} \rho(z_1, z_2)$$

На практике вычисляют отношение пары функционалов, чтобы учесть, как межкластерные, так и внутрикластерные расстояния:

$$\frac{\Phi_0}{\Phi_1} \to min, \qquad \frac{\Phi_1}{\Phi_0} \to max$$

Будем в дальнейшем считать, что проведена дискретизация компонент векторов $\boldsymbol{a} = (a_1, a_2, \dots, a_D) \in \boldsymbol{V}$ на $\boldsymbol{Z}$ уровней. Это позволяет использовать мощный алгебраический аппарат конечных полей и строить кодовые конструкции, описанные в следующем разделе.

3. Описание суффиксной конструкции кодов в модульной метрике для решения задачи классификации методом KNN

Пусть задано множество $\boldsymbol{V}$ состоящее из N векторов $\boldsymbol{a} = (a_1, a_2, \dots, a_D) \in \mathbf{V}, \boldsymbol{b} = (b_1, b_2, \dots, b_D) \in \boldsymbol{V}$. Каждый вектор множества $\boldsymbol{V}$ состоит из D компонентов над

подмножеством целых чисел $a_i \in \{0,1,2,\dots Z-1\} \equiv \boldsymbol{Z}$, $b_i \in \{0,1,2,\dots Z-1\} \equiv \boldsymbol{Z}$. Расстояние в модульной метрике между векторами $\boldsymbol{a}$ и $\boldsymbol{b}$ определяется по формуле

$$d_M(\boldsymbol{a},\boldsymbol{b}) = \sum_{i=1}^{D} |a_i - b_i|$$

Определим отношение $\boldsymbol{a} \gg \boldsymbol{b}: \{a_i \geq b_i \mid 1 \leq i \leq D\}$ и отношение $\boldsymbol{a} \ll \boldsymbol{b}: \{a_i \leq b_i \mid 1 \leq i \leq D\}$. Если $d_M(\boldsymbol{a},\boldsymbol{b}) = t$ и $\boldsymbol{a} \gg \boldsymbol{b}$ ($\boldsymbol{a} \ll \boldsymbol{b}$) будем говорить, что вектор $\boldsymbol{a}$ получен из вектора $\boldsymbol{b}$ путем t увеличивающих (уменьшающих) вес ошибок в модульной метрике. Такие ошибки будем называть однонаправленными ошибками.

Будем называть подмножество векторов $\mathbb{C} \subset \boldsymbol{V}$ кодом, исправляющим t увеличивающих (уменьшающих) вес ошибок в модульной метрике если выполняются условия $\forall\, \boldsymbol{a} \in \mathbb{C},\ \forall\, \boldsymbol{b} \in \mathbb{C}, \nexists\, \boldsymbol{c} \in \boldsymbol{V}: \boldsymbol{c} \gg \boldsymbol{a}, \boldsymbol{c} \gg \boldsymbol{b}\ (\boldsymbol{c} \ll \boldsymbol{a}, \boldsymbol{c} \ll \boldsymbol{b}), d_M(\boldsymbol{a},\boldsymbol{c}) \leq t, d_M(\boldsymbol{b},\boldsymbol{c}) \leq t$

Пусть задано конечное поле из Q элементов $GF(\mathrm{Q})$ и выполняется условие $\mathrm{Q} > Z$, т.е. мультипликативная группа поля должна содержать не менее Z элементов. Пусть заданы множество различных ненулевых элементов поля $L = \{l_1, l_2, \dots, l_D\} \subset GF(\mathrm{Q})$ которое будем называть множеством локаторов и множество различных ненулевых элементов поля $S = \{s_1, s_2, \dots, s_T\} \subset GF(\mathrm{Q})$ которое будем называть множеством суффиксов. Для $1 \leq t \leq \mathrm{T}$ будем обозначать подмножества мощности $t: S_t = \{s_1, s_2, \dots, s_t\} \subseteq \{s_1, s_2, \dots, s_T\}$. Будем считать, что $L \cap S = \emptyset$. Определим отображение вектора $\boldsymbol{u} = (u_1, u_2, \dots, u_D) \in \boldsymbol{V}$ в локаторный полином $u(x)$

$$u(x) = \mathcal{F}(\boldsymbol{u}) \triangleq \prod_{i=1}^{D} \left(1 - \frac{x}{l_i}\right)^{u_i}$$

Для подмножества суффиксов $S_t = \{s_1, s_2, \dots, s_t\}$ определим суффиксный полином

$$s_t(x) \triangleq x \prod_{i=1}^{t} \left(1 - \frac{x}{s_i}\right)$$

Обозначим $supp\big(s(x)\big)$ – множество корней полинома $s(x)$. Из определения полиномов $u(x)$ и $s_t(x)$ следует, что $supp(u(x)) \cap supp\big(s_t(x)\big) = \emptyset$. Обозначим $f(x)$ фиксированный полином от формальной переменной x с коэффициентами над полем $GF(\mathrm{Q})$, который является остатком некоторого локаторного полинома $u(x)$ с коэффициентами над полем $GF(\mathrm{Q})$ по модулю полинома $s_t(x)$.

$$f(x) = u(x)\ mod\ s_t(x)$$

В работе [3] доказываются две теоремы, важные для дальнейших рассуждений.

Теорема 1.

Код $\mathbb{C}_{\boldsymbol{f}} = \big\{\boldsymbol{u} \in \boldsymbol{V}: u(x) \equiv f(x)\ mod\ s_t(x), f(x) \in \boldsymbol{\mathcal{F}_Q}/s_t(x)\big\}$ исправляет не менее t однонаправленных ошибок в модульной метрике.

Теорема 2.

Код $\boldsymbol{\mathcal{B}_\delta}$

$$\boldsymbol{\mathcal{B}_\delta} = \left\{\boldsymbol{c} \in \mathbb{C}: \sum_{i=1}^{D} |\mathrm{c}_i| \ \equiv\ \delta\ mod\ (2t+1), 0 \leq\ \delta \leq 2t\right\}$$

исправляет не менее t ошибок в модульной метрике.

Из определения суффиксного полинома $s_t(x)$, локаторного полинома $u(x)$, а также китайской теоремы об остатках следует, что соответствующий $f(x)$ однозначно определяется

t вычетами полинома $u(x)$ по модулю полиномов $(1 - xs_i^{-1})$, $1 \le i \le t$. Каждый такой вычет является ненулевым элементом поля $GF(\mathrm{Q})$. Нулевой вычет невозможен, поскольку что $L \cap S = \emptyset$. Будем обозначать для полинома $u(x)$ такие ненулевые вычеты $u_i^* \equiv u(x) \bmod (1 - xs_i^{-1})$, $1 \le i \le t$. Таким образом получаем отображение $\mathfrak{J}$ произвольного вектора $\boldsymbol{u} = (u_1, u_2, \dots, u_D) \in \boldsymbol{V}$ в вектор ненулевых вычетов $\boldsymbol{u}^* = (u_1^*, u_2^*, \dots, u_t^*)$.

$$\mathfrak{J}(\mathbf{u}) \triangleq (u_1^*, u_2^*, \dots, u_t^*)$$

Из утверждения Теоремы 1 следует, что вектору ненулевых вычетов $\boldsymbol{u}^* = (u_1^*, u_2^*, \dots, u_t^*)$ соответствует либо свой код $\mathbb{C}$**,** исправляющий не менее t однонаправленных ошибок в модульной метрике, поскольку каждый $f(x) \in \boldsymbol{\mathcal{F}_Q}/s_t(x)$ однозначно описывается своим вектором вычетов, либо вектору ненулевых вычетов $\boldsymbol{u}^* = (u_1^*, u_2^*, \dots, u_t^*)$ соответствует пустое множество $\emptyset$. Аналогично из утверждения Теоремы 2 следует, что каждому вектору ненулевых вычетов $\boldsymbol{u}^* = (u_1^*, u_2^*, \dots, u_t^*)$ и целому числу $0 \le \delta \le 2t$ соответствует либо свой код $\boldsymbol{\mathcal{B}}$**,** исправляющий не менее t произвольных ошибок в модульной метрике, либо пустое множество $\emptyset$. Сформулируем данные утверждения в виде Следствий.

Следствие из Теоремы 1.

Пусть задан вектор $(u_1^*, u_2^*, \dots, u_t^*), u_i^* \in GF(\mathrm{Q}), u_i^* \ne 0,\ 1 \le i \le t$. Тогда либо $\mathbb{C} = \{\boldsymbol{u} \in \boldsymbol{V}: \mathfrak{J}(\mathbf{u}) \triangleq (u_1^*, u_2^*, \dots, u_t^*)\}$ исправляет не менее t однонаправленных ошибок в модульной метрике, либо $\mathbb{C} \equiv \emptyset$.

Следствие из Теоремы 2.

Пусть задан вектор $(u_1^*, u_2^*, \dots, u_t^*), u_i^* \in GF(\mathrm{Q}), u_i^* \ne 0,\ 1 \le i \le t$ и задано целое число $0 \le \delta \le 2t$. Тогда либо $\boldsymbol{\mathcal{B}} = \{\boldsymbol{u} \in \boldsymbol{V}: \mathfrak{J}(\mathbf{u}) \triangleq (u_1^*, u_2^*, \dots, u_t^*),\ \sum_{i=1}^{D} |u_i| \equiv \delta \bmod (2t + 1)\}$ исправляет не менее t произвольных ошибок в модульной метрике, либо $\boldsymbol{\mathcal{B}} \equiv \emptyset$.

Сформулируем конструкцию для метода KNN с использованием суффиксных кодов в модульной метрике, согласно Следствию из Теоремы 2.

4. Использование суффиксной конструкции кодов в модульной метрике для обработки тестовой выборки

Пусть N объем тестовой выборки $\boldsymbol{V}$. Выборка содержит C различных классов объектов. Каждый вектор $\boldsymbol{V}$ состоит из D компонентов над подмножеством целых чисел $\{0,1,2, \dots Z - 1\} \equiv \boldsymbol{Z}$**.** Такую выборку будем обозначать (N, D, Z, C). Зададим максимальную величину ошибок T, которую будет исправлять кодовая конструкция. Как вариант, можно считать, что $T \approx max\{D, Z\}$. Пусть задано $GF(Q)$ и выполняется неравенство $Q > max\{D + T, Z\}$. Данное условие позволяет обеспечить формирование множества локаторов $L = \{l_1, l_2, \dots, l_D\} \subset GF(\mathrm{Q})$ и суффиксов $S = \{s_1, s_2, \dots, s_T\}$ для которых выполняется условие $L \cap S = \emptyset$.

Будем строить конструкцию кодов, для исправления t ошибок в модульной метрике, где $1 \le t \le \mathrm{T}$. Поставим в соответствие каждой из D позиций векторов $\boldsymbol{v} = (v_1, v_2, \dots, v_D) \in \boldsymbol{V}(N, D, Z, C)$ ненулевой элемент поля из множества локаторов $L = \{l_1, l_2, \dots, l_D\} \subset GF(\mathrm{Q})$.

Для любого $1 \le t \le \mathrm{T}$ зададим отображение $W(\boldsymbol{v}, t)$ вектора $\boldsymbol{v} = (v_1, v_2, \dots, v_D)$ в локаторный полином степени t от формальной переменной $\boldsymbol{x}$ над полем $GF(Q)$ по формуле

$$W(\boldsymbol{v}, t) = \prod_{i=1}^{D} \left(1 - \frac{x}{l_i}\right)^{v_i} mod \left(x \prod_{i=1}^{t} \left(1 - \frac{x}{s_i}\right)\right)$$

Значения s_i выбираются из множества суффиксов $S_t = \{s_1, s_2, \dots, s_t\} \subset \{s_1, s_2, \dots, s_T\}$. Заметим, что младший коэффициент $\boldsymbol{W}(\boldsymbol{v}, t)$ всегда равен 1 для любых значений $t > 1$ и любого вектора $\boldsymbol{v} = (v_1, v_2, \dots, v_D) \in \boldsymbol{V}$. Число различных полиномов $\boldsymbol{W}(\boldsymbol{v}, t)$ не превосходит величины $(\boldsymbol{Q} - \boldsymbol{1})^{\boldsymbol{t}}$.

5. Обработка обучающей выборки путем иерархической кластеризации

Процедура обработки множества слов $\boldsymbol{V}(N,D,Z,C)$ производится последовательно для подмножеств, соответствующих каждому классу из $\boldsymbol{Y}=\{1,2,\dots C\}$. Для дальнейших рассуждений выберем класс номер 1 из множества таких классов $\boldsymbol{Y}=\{1,2,\dots C\}$. Обработка $\boldsymbol{V}(N,D,Z,C)$ для каждого класса осуществляется последовательно для всех значений $1\le t\le T$, начиная с минимального.

Для векторов данного класса из множества слов $\boldsymbol{V}(N,D,Z,C)$ и заданного значения t рассмотрим полином $\boldsymbol{W}(\boldsymbol{a},\boldsymbol{t})$ для каждого $\boldsymbol{a}=(a_1,a_2,\dots,a_D)\in\boldsymbol{V}$ и заданного множества суффиксов $S_t=\{s_1,s_2,\dots,s_t\}\subset\{s_1,s_2,\dots,s_T\}$. Выберем наиболее часто встречающийся вариант. Обозначим число векторов такого варианта n_{1t}. Обозначим такой полином $\boldsymbol{V}(\boldsymbol{x},\boldsymbol{t},\boldsymbol{1})$. Согласно Следствию из Теоремы 2, подмножество слов $\boldsymbol{V}(N,D,Z,C)$, имеющих одинаковый $\boldsymbol{V}(\boldsymbol{x},\boldsymbol{t},\boldsymbol{1})$ является кодом, с исправлением t однонаправленных ошибок в модульной метрике. Будем обозначать такой код $\boldsymbol{V}_1^*(n_{1t},D,Z,C,t)$.

Проведем процедуру исправления t ошибок во всех векторах $\boldsymbol{V}(N,D,Z,C)$, относящихся к первому классу, чей полином не совпадает с $\boldsymbol{V}(\boldsymbol{x},\boldsymbol{t},\boldsymbol{1})$, т.е. найдем ошибки, которые показывают, насколько слово кода $\boldsymbol{V}_1^*(n_{1t},D,Z,C,t)$ отличается от слова $\boldsymbol{a}=(a_1,a_2,\dots,a_D)\in\boldsymbol{V}(N,D,Z,C)$. Для этого необходимо последовательно предположить, что число t произошедших ошибок распределилось между ошибками, увеличивающими вес и ошибками, уменьшающими вес, т.е. рассмотреть $2t+1$ вариантов.

Производя последовательно $2t+1$ декодирований, получаем различные варианты кодового вектора, ближайшего к анализируемому слову $\boldsymbol{a}=(a_1,a_2,\dots,a_D)\in\boldsymbol{V}$, не являющегося кодовым. Из полученных вариантов выбираем то кодовое слово, которое находится на минимальном расстоянии к анализируемому вектору $\boldsymbol{a}=(a_1,a_2,\dots,a_D)\in\mathbf{V}(N,D,Z,C)$. В результате вместо вектора $\boldsymbol{a}=(a_1,a_2,\dots,a_D)\in\mathbf{V}(N,D,Z,C)$ для текущего значения $1\le t\le$ T получаем вектор $\boldsymbol{a}^*=(a_1^*,a_2^*,\dots,a_D^*)\in\mathbf{V}_1^*(n_{1t},D,Z,C,t)$. Заметим, что

$$d_M(\boldsymbol{a}^*,\boldsymbol{a})=\sum_{i=1}^{D}|a_i^*-a_i|\le t$$

В одно слово кода $\boldsymbol{V}_1^*(n_{1t},D,Z,C,t)$ может быть декодировано несколько слов $\mathbf{V}(N,D,Z,C)$, относящихся к классу 1. Для каждого слова $\boldsymbol{V}_1^*(n_{1t},D,Z,C,t)$ фиксируется сколько слов $\boldsymbol{V}(N,D,Z,C)$ из класса 1 было декодировано в данное слово. Будем называть такое число мощностью слова $\boldsymbol{a}^*=(a_1^*,a_2^*,\dots,a_D^*)\in\boldsymbol{V}_1^*(n_{1t},D,Z,C,t)$. Некоторые слова кода, соответствующего полиному $\boldsymbol{V}(\boldsymbol{x},\boldsymbol{t},\boldsymbol{1})$ могут иметь нулевую мощность. Это означает, что в них не было декодировано ни одного слова из $\boldsymbol{V}(N,D,Z,C)$, относящегося к классу 1.

Отметим, что для каждого значения $1\le t\le$ T и для каждого слова кодов $\boldsymbol{V}_1^*(n_{1t},D,Z,C,t)$ определяется свое значение мощности. Множества суффиксов удовлетворяет условию

$$S_1\subset\cdots\subset S_{t-1}\subset S_t\subset S_{t+1}\subset\cdots\subset S_T$$

Из данного условия следует, что код, исправляющий $t+1$ ошибок, является подмножеством кода с исправлением t ошибок, т.е. выполняется условие $\boldsymbol{V}_1^*(n_{1t},D,Z,C,t)\supset\boldsymbol{V}_1^*\left(n_{1,t+1},D,Z,C,t+1\right)$, то каждое слово кода $\boldsymbol{V}_1^*\left(n_{1,t+1},D,Z,C,t+1\right)$ имеет список из $t+1$ мощностей для всех кодов, в которые данное слово входит.

Поскольку множество слов $\boldsymbol{V}_1^*(n_{1t},D,Z,C,t)$ является кодом, исправляющим однонаправленные t ошибок, возможна ситуация, когда один и тот же вектор $(a_1,a_2,\dots,a_D)\in\boldsymbol{V}(N,D,Z,C)$ декодируется сразу в несколько слов $\boldsymbol{V}_1^*(n_{1t},D,Z,C,t)$. Это возможно, если произошли разнонаправленные ошибки. В этом случае будем выбирать то кодовое слово из $\boldsymbol{V}_1^*(n_{1t},D,Z,C,t)$, расстояние до которого меньше, а если несколько слов кода находятся на одинаковом расстоянии – выбирать кодовое слово с большей мощностью.

Процедуру, описанную выше, проведем для всех классов из множества таких классов $\boldsymbol{Y} = \{1, 2, \dots C\}$. В результате для каждого значения $1 \le t \le T$ получим множества $\boldsymbol{V}_1^*(n_{1t}, D, Z, C, t)$, $\boldsymbol{V}_2^*(n_{2t}, D, Z, C, t)$, … $\boldsymbol{V}_C^*(n_{Ct}, D, Z, C, t)$. Объединение данных множеств дает адаптированную выборку $\boldsymbol{V}^*(N^*, D, Z, C, t)$. Другими словами, выполняется условие

$$\boldsymbol{V}^*(N^*, D, Z, C, t) = \bigcup_{i=1}^{C} \boldsymbol{V}_i^*(n_{it}, D, Z, C, t)$$

Заметим, что для числа слов полученных кодов с ненулевой мощностью для каждого $1 \le t \le T$ выполняется неравенство.

$$N \ge \sum_{i=1}^{C} n_{it} = N^*$$

Выбор наиболее часто встречающегося варианта полинома $\boldsymbol{V}(\boldsymbol{x}, \boldsymbol{t}, \boldsymbol{1})$ эмпирически определяет наиболее подходящий код, как код, для которого надо исправлять ошибки в меньшем числе слов. Другими словами, у такого кода максимальное число кодовых слов совпало с числом векторов $\boldsymbol{V}(N, D, Z, C)$, относящимся к первому классу объектов. Выбор может произведен и по другим критериям. Например, по минимальному расстоянию от выбранных кодовых слов до ближайших векторов $\boldsymbol{V}(N, D, Z, C)$.

В дальнейшем будем для определения k ближайших соседей для вектора $\boldsymbol{f} = (f_1, f_2, \dots, f_D)$ использовать $\boldsymbol{V}_1^*(n_{1t}, D, Z, C, t)$, $\boldsymbol{V}_2^*(n_{2t}, D, Z, C, t)$, … $\boldsymbol{V}_C^*(n_{Ct}, D, Z, C, t)$ для каждого значения $1 \le t \le T$, последовательно переходя от меньших значений t к большим. Признаком остановки будет являться получение кодовых слов с суммарной мощностью не менее *k*.

6. Поиск ближайших k соседей к новому объекту

Выбираем начальное значение $t = 1$ и для каждого класса из множества классов $\mathbf{Y} = \{1,2, \dots C\}$, проведем декодирование вектора $\boldsymbol{f} = (f_1, f_2, \dots, f_D)$ в ближайшие кодовые слова кодов $\boldsymbol{V}_1^*(n_{1t}, D, Z, C, t)$, $\boldsymbol{V}_2^*(n_{2t}, D, Z, C, t)$, … $\boldsymbol{V}_C^*(n_{Ct}, D, Z, C, t)$.

Полученные слова каждого кода имеют свою мощность (т.е. число слов исходной обучающей выборки, которые были декодированы в данное слово). Если сумма мощностей таких слов больше или равна k, то выбранный параметр t достаточен. Если сумма мощностей меньше k – необходимо увеличить параметр до t+1 и повторить действие алгоритма для кодов $\boldsymbol{V}_1^*(n_{1,t+1}, D, Z, C, t+1)$, $\boldsymbol{V}_2^*(n_{2,t+1}, D, Z, C, t+1)$, … $\boldsymbol{V}_C^*(n_{C,t+1}, D, Z, C, t+1)$.

Для классифицируемого объекта $\boldsymbol{f} = (f_1, f_2, \dots, f_D)$ выбирается тот класс из множества $\boldsymbol{Y} = \{1,2, \dots C\}$, для которого мощность полученного кодового слова больше. Если два слова разных классов имеют одинаковую мощность – то вычисляется расстояние в модульной метрике от вектора $\boldsymbol{f} = (f_1, f_2, \dots, f_D)$ до полученных кодовых слов, имеющих одинаковую мощность, и выбирается тот класс, расстояние до которого меньше.

В результате принятия решения по классу нового объекта, увеличивается мощность того кодового слова, в которое был декодирован классифицируемый вектор $\boldsymbol{f} = (f_1, f_2, \dots, f_D)$. Далее алгоритм поиска ближайших k соседей повторяется для классификации нового вектора и так далее.

7. Заключение

Предложенная конструкция по использованию суффиксных кодов в модульной метрике для решения задачи иерархической кластеризации и задачи поиска k соседей позволяет реализовать решение данных задач с полиномиальной сложностью от размерности обучающей выборки. Конструкция, при выборе достаточно большого значения T не требует повторной процедуры обработки обучающей выборки при переходе к большему значению k при решении задачи поиска k соседей.

Список литературы / References

[1]. В. А. Давыдов. Использование кодов в модульной метрике для решения задачи KNN. (в публикации).

[2]. В. А. Давыдов. Коды, исправляющие ошибки в модульной метрике, метрике Ли и ошибки оператора // Пробл. передачи информ. 1993. Глава 29:3. С. 209–217

[3]. V. Davydov, N. Zeulin, I. Pastushok, A. Turlikov. Coding Scheme for High-Order QAM Modulations in the Manhattan Metric // IEEE Communications Letters. 2020. Vol. 24. N. 11. P. 2387–2391.

[4]. T. Cover, P. Hart. Nearest neighbor pattern classification // IEEE Transactions on Information Theory. 1967. Vol. 13. N. 1. P. 21–27.

[5]. Fix. E, Hodges, J.L. Discriminatory analysis. Nonparametric discrimination; consistency properties // USAF School of Aviation Medicine. 1951. Technical Report 4.

[6]. P. Indyk, R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality // Proceedings of the Symposium on Theory of Computing. 1998.

[7]. E. Kushilevitz, R. Ostrovsky, Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces // Proceedings of the Thirtieth ACM Symposium on Theory of Computing. 1998. P. 614– 623.

[8]. J. Buhler, M. Tompa. Finding motifs using random projections // Proceedings of the Annual International Conference on Computational Molecular Biology. 2001.

[9]. J. Buhler. Provably sensitive indexing strategies for biosequence similarity search // Proceedings of the Annual International Conference on Computational Molecular Biology. 2002.

[10]. N. C. Jones, P. A. Pevzner. An Introduction to Bioinformatics Algorithms // The MIT Press Cambridge. 2004.

[11]. M. Namratha. IOSR Journal of Computer Engineering. 2012. Vol. 4(6). P 23–30.

[12]. A. V. Kumar, J. C. Selvaraj. Journal of Recent Research and Applied Studies. 2016. Vo_103.

[13]. M. Omran, A. Engelbrecht, A. A. Salman. Intelligent Data Analysis. 2007. Vol. 11(6). P. 583–605.

[14]. M. Wegmann, D. Zipperling, J. Hillenbrand, J. Fleischer. A review of systematic selection of clustering algorithms and their evaluation. 2021.

[15]. J. Gu. Journal of Physics: Conference Series. 2021. Vol. 1.

[16]. T. W. Liao. Pattern Recognition. 2005. Vol. 38(11). P. 1857–1874.

[17]. T. Gupta, S. Panda. International Journal of Engineering & Technology. 2018. Vol. 7(4). P. 4766–4768.

Информация об авторах / Information about authors

Александр Рауилович ШАРАПОВ – является аспирантом московского института электроники и математики им. А.Н. Тихонова, департамент электронной инженерии. Темой исследования является «Разработка метода анализа данных с использованием кодов, исправляющих ошибки в метрике l1». Сфера научных интересов: статистика, анализ данных, кодирование информации, сети и телекоммуникации.

Alexander Rauilovich SHARAPOV is a postgraduate student at the Moscow Institute of Electronics and Mathematics named after A.N. Tikhonov, Department of Electronic Engineering. The topic of his research is "Development of a method for analyzing data using codes that correct errors in the l1 metric". His research interests include statistics, data analysis, information coding, networks and telecommunications.

Вячеслав Анатольевич ДАВЫДОВ – кандидат технических и экономических наук. Он является приглашенным преподавателем московского института электроники и математики им. А.Н. Тихонова, кафедра информационной безопасности киберфизических систем. Сфера научных интересов: алгебраическая и комбинаторная теория кодирования, теория активных систем, технология блокчейн.

Vyacheslav Anatolyevich DAVYDOV – Cand. Sci. (Tech., Econ.). He is a visiting lecturer at the Moscow Institute of Electronics and Mathematics named after A.N. Tikhonov, Department of Information Security of Cyber-Physical Systems. His research interests include algebraic and combinatorial coding theory, active systems theory, blockchain technology.

DOI: 10.15514/ISPRAS-2025-37(5)-3

Generating Compact Residue Number Systems Bases

V.V. Lutsenko, ORCID: 0000-0003-4648-8286 <officialvladlutsenko@gmail.com>
M.G. Babenko, ORCID: 0000-0001-7066-0061 <mgbabenko@ncfu.ru>
North-Caucasus Federal University, Stavropol,
1, Pushkin st., Stavropol, 355017, Russia.

Abstract. Modern computational tasks involving large-number processing demand not only high precision but also significant operational speed. In this context, the residue number system provides an effective approach for parallel processing of large numbers, with applications in cryptography, signal processing, and artificial neural networks. The primary task in defining such a system is determining its basis. This paper presents an algorithm for generating compact residue number system bases based on the Diemitko theorem. The proposed algorithm generates bases 15.5% faster on average than Pseudo-Mersenne-based construction and 75.7% faster than the general filtering method. Comparative analysis demonstrates that using compact bases delivers an average 12% acceleration in modular operations compared to special moduli sets.

Keywords: residue number system; high-performance computing; special sets of moduli; generation of prime numbers; cryptography.

For citation: Lutsenko V.V., Babenko M.G. Generating compact residue number systems bases. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025. pp. 43-52. DOI: 10.15514/ISPRAS-2025-37(5)-3.

Acknowledgements. The research was supported by the Russian Science Foundation Grant No. 25-71-30007, https://rscf.ru/en/project/25-71-30007/.

Генерация компактных базисов системы остаточных классов

В.В. Луценко, ORCID: 0000-0003-4648-8286 <officialvladlutsenko@gmail.com>
М.Г. Бабенко, ORCID: 0000-0001-7066-0061 <mgbabenko@ncfu.ru>

Северо-Кавказский федеральный университет,
Россия, 355017, г. Ставрополь, ул. Пушкина, д. 1.

Аннотация. Современные вычислительные задачи, связанные с обработкой больших чисел, требуют не только высокой точности, но и значительной скорости операций. В данном контексте применение системы остаточных классов предлагает подход к параллельной обработке больших чисел, который применяется в криптографии, обработке сигналов и искусственных нейронных сетях. Ключевой задачей при построении системы остаточных классов является определение её базиса. В статье представлен алгоритм генерации компактных базисов системы остаточных классов, основанный на теореме Диемитко. Предложенный алгоритм генерирует базисы в среднем на 15,5% быстрее, чем построение базисов на основе псевдо-Мерсенновских чисел, и на 75,7% быстрее, чем метод общей фильтрации. Проведённый сравнительный анализ показал, что использование компактных базисов обеспечивает в среднем 12% ускорение модульных операций по сравнению со специальными наборами модулей.

Ключевые слова: система остаточных классов; высокопроизводительные вычисления; специальные наборы модулей; генерация простых чисел; криптография.

Для цитирования: Луценко В.В., Бабенко М.Г. Генерация компактных базисов системы остаточных классов. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 43–52 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)–3.

Благодарности. Исследование выполнено за счет гранта Российского научного фонда № 25-71-30007, https://rscf.ru/project/25-71-30007/.

1. Introduction

Modern computational problems involving the processing of large numbers require not only high accuracy but also significant speed of operations. In this context, unconventional arithmetic offers innovative approaches that optimize computation in various areas such as cryptography, signal processing and theoretical computer science. One of the key tools in this area is the Residual Number System (RNS), which dates back to the 1950s and is based on the Chinese Remainder Theorem (CRT) [1]. RNS is an alternative way of representing numbers based on modular arithmetic. Instead of dealing with numbers in a positional representation, RNS decomposes them into a set of residues obtained by division by pairwise prime numbers, called the RNS basis [2]. The main advantage of RNS is that the addition and multiplication operations are performed in parallel on each residue, which greatly speeds up the computation. Despite the cost of the inverse transformation, which in the worst case depends quadratically on the size of the basis, computations involving addition and multiplication become extremely performant. However, inverse transformation, division and comparison of numbers in RNS remain computationally challenging problems, but recent works propose efficient algorithms for these tasks [3-5].

RNS has been applied in signal processing [6], cryptography [7], and neural networks [8]. The research presented in this paper is relevant to a wide range of applications related to large number processing, including cryptographic systems since the 1990s [9], such as RSA, DH, ECC [10-11], as well as pairing methods, Euclidean lattice-based algorithms and homomorphic protocols [12].

In cryptography, where arithmetic operations are performed modulo large numbers that are often prime, the application of RNS becomes more challenging due to the need to perform modulo taking, which has led to active research aimed at selecting optimal bases to improve implementation efficiency [13-14]. RNS is also of particular interest for defense against error injection attacks, as the introduction of redundant elements at the basis level allows error detection mechanisms to be

organized [15]. In addition, the random choice of basis provides a different representation of data at each computation, which complicates the analysis of possible information leaks. Thus, the choice of basis is the first and most important task of RNS.

In this paper we present a method for generating RNS bases based on Diemitko's theorem. This approach allows us to obtain bases satisfying the compactness condition.

The article is structured as follows. Following the introduction, Section 2 covers the fundamentals of RNS. Section 3 reviews related works. Section 4 presents the proposed algorithm for generating compact RNS bases. Section 5 then evaluates the algorithm's performance. Finally, the key findings are summarized in the conclusion.

2. Residue Number System

RNS is based on the widely known CRT [16]. RNS argues that, knowing the smallest non-negative residues from dividing an integer X by the integer moduli $p_1, p_2, \dots, p_n$ it is possible to uniquely determine the residue from dividing X by the product of these moduli, provided that the moduli are pairwise coprime. RNS, unlike classical b-ary number systems, is not defined by a single fixed base, but by a set of moduli $\{p_1, p_2, \dots, p_n\}$ such that $gcd(p_i, p_j) = 1$ for all $i, j \in 1,2, \dots n, i \neq j$, where $gcd(\ \ )$ is the greatest common divisor. The product of these moduli $P = \prod_{i=1}^{n} p_i$ determines the dynamic range of the RNS. An integer $X \in [0, P)$ is represented as a vector composed of the smallest non-negative residues obtained by dividing X by p_i:

$$X = (x_1, x_2, \dots, x_n). \quad (1)$$

where $x_i = X(mod\ p_i)$, which is also denoted by $x_i = |X|_{p_i}$.

Consider RNS with the basis $\{4,5,7\}$. In this basis, we can mutually uniquely represent the numbers from the half-interval $[0; 140)$, since $P = 140$.

Table 1 shows the correspondences of numbers from the positional number system and the RNS.

Table 1. Representation of Numbers for RNS with the Basis {4,5,7}.

$0 \overset{RNS}{\longrightarrow} (0,0,0)$	$1 \overset{RNS}{\longrightarrow} (1,1,1)$	$2 \overset{RNS}{\longrightarrow} (2,2,2)$	$3 \overset{RNS}{\longrightarrow} (3,3,3)$
$4 \overset{RNS}{\longrightarrow} (0,4,4)$	$5 \overset{RNS}{\longrightarrow} (1,0,5)$	$6 \overset{RNS}{\longrightarrow} (2,1,6)$	$7 \overset{RNS}{\longrightarrow} (3,2,0)$
$8 \overset{RNS}{\longrightarrow} (0,3,1)$	$9 \overset{RNS}{\longrightarrow} (1,4,2)$	$10 \overset{RNS}{\longrightarrow} (2,0,3)$	$11 \overset{RNS}{\longrightarrow} (3,1,4)$

RNS defines basic operations on numbers, which are divided into two groups. The operations of the first group, which are sometimes called modular, include addition and subtraction of numbers without the possibility of determining the sign of the result, as well as multiplication. Such operations are performed component-wise on remainders, i.e. without forming carryovers between them. Let the numbers X, Y and R be represented as $(x_1, x_2, \dots, x_n)$, $(y_1, y_2, \dots, y_n)$ and $(r_1, r_2, \dots, r_n)$, respectively. Then for any modular operation $\circ$ we have

$$X \circ Y = (r_1, r_2, \dots, r_n), \quad (2)$$

where $r_i = |x_i \circ y_i|_{p_i}$.

Thus, the i-th digit of the result in RNS, r_i, is defined only in terms of $|x_i \circ y_i|_{p_i}$ and does not depend on any other digit r_j. This allows the realization of carry-free, high-speed (parallel) computer arithmetic and makes RNS an attractive number system for use in resource-intensive applications, especially those involving the processing of large numbers. It also provides high computational reliability since an error in the *i*-th digit has no effect on other digits and therefore can be efficiently localized and eliminated [15]. In turn, for operations of the second group, often called non-modular, it is not enough to know the values of individual residues and requires an estimate of the magnitude of numbers: the result of such an operation is either not a number in RNS at all, or the value of each

of its digits (residue) is not only a function of the values of the corresponding digits of the operands, but depends on the magnitude of these operands.

Algorithm 1 presents a method for performing modular operations in RNS.

Algorithm 1: Modular operations in RNS.

Input: $\{p_1, p_2, \dots, p_n\}, (x_1, x_2, \dots, x_n), (y_1, y_2, \dots, y_n), \circ \in \{+, -, \times\}$
Output: $R = (r_1, r_2, \dots, r_n)$
1. **for** $i = 1, i \le n, i++$ **do**:
1.2 $r_i = |x_i \circ y_i|_{p_i}$

Here is an example of addition in RNS.

Example 1 (Addition in RNS). Let us add two numbers $X = (2, 0, 3)$ and $Y = (3, 0, 1)$ in the basis {4,5,7}. Use (2) for addition:

$$X + Y = (|2 + 3|_4, |0 + 0|_5, |3 + 1|_7) = (1,0,4)$$

Hence $R = (1,0,4)$. Which is true since $25 \xrightarrow{RNS} (1, 0, 4)$.

3. Related Works

The choice of modulo set is very important to achieve a suitable RNS implementation. The modulo set affects the whole RNS architecture [17]. Special sets of moduli are widely used [18]. Table 2 presents the most well-known special sets of moduli.

Table 2. Special sets of moduli.

Number	Set	Year
1	$\{2^n - 1, 2^n, 2^n + 1\}$	1967
2	$\{2n - 1, 2n, 2n + 1\}$	1995
3	$\{2^{2n} + 1, 2^n + 1, 2^n - 1\}$	1997
4	$\{2^n - 1, 2^n, 2^{n-1} - 1\}$	1998
5	$\{2^n - 1, 2^n, 2^{n+1} - 1\}$	1999
6	$\{2^n - 1, 2^n, 2^{2n+1} - 1\}$	2008
7	$\{2^n - 1, 2^n, 2^{2n} + 1\}$	2008
8	$\{2^\alpha, 2^\beta - 1, 2^\beta + 1\}$	2008
9	$\{3^n - 2, 3^n - 1, 3^n\}$	2007
10	$\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} + 1\}$	1999
11	$\{2^n - 1, 2^n, 2^n + 1, 2^{n+1} - 1\}$	2000
12	$\{2^n - 1, 2^n, 2^n + 1, 2^{2n} - 1\}$	2003
13	$\{2^n - 1, 2^n + 1, 2^n - 3, 2^n + 3\}$	2004
14	$\{2^n - 1, 2^n + 1, 2^{2n} - 2, 2^{2n+1} - 3\}$	2008
15	$\{2^n - 1, 2^n + 1, 2^n, 2^{2n} + 1\}$	2009
16	$\{2^n - 1, 2^n, 2^n + 1, 2^{2n+1} - 1\}$	2009
17	$\{2^n - 1, 2^n + 1, 2^{2n}, 2^{2n+1} - 1\}$	2009
18	$\{2^k, 2^n - 1, 2^n + 1, 2^{n+1} + 1\}$	2014
19	$\{2^k, 2^n - 1, 2^n + 1, 2^{n+1} - 1\}$	2014
20	$\{2^n - 1, 2^n, 2^n + 1, 2^n - 2^{(n+1)/2} + 1, 2^n + 2^{(n+1)/2} + 1\}$	2005
21	$\{2^n - 1, 2^n, 2^n + 1, 2^{n-1} - 1, 2^{n+1} - 1\}$	2007
22	$\{2^{n/2} - 1, 2^n, 2^{n/2} + 1, 2^n + 1, 2^{2n-1} - 1\}$	2009
23	$\{2^n - 1, 2^n, 2^n + 1, 2^n - 2^{(n+1)/2} + 1, 2^n + 2^{(n+1)/2} + 1, 2^{n\pm1} + 1\}$	2012
24	$\{2^n - 1, 2^{n+\beta}, 2^n + 1, 2^n - 2^{(n+1)/2} + 1, 2^n + 2^{(n+1)/2} + 1, 2^{n\pm1} + 1\}$	2012
25	$\{2^{n+\beta}, 2^n - 1, 2^n + 1, 2^n - k_1, 2^n + k_1, \dots, 2^n - k_f, 2^n + k_f\}$	2018

In the work [19], the sets of modules from Table 2 were investigated. As a result of the experiments, the set of modules $\{2^n - 1, 2^n, 2^n + 1\}$ turned out to be the most effective.

In cryptography, it is possible to use general-form moduli sets. By increasing the number of moduli, a higher degree of parallelism can be achieved. The literature often describes methods for generating RNS bases based on Pseudo-Mersenne numbers [20]. The paper [14] proposes a filtering method for constructing a very large RNS basis. However, these approaches do not account for the compactness condition of the RNS basis.

Definition 1. A set of moduli $\{p_1, p_2, \dots, p_n\}$, where $p_1 < p_2 < \cdots < p_n$, is compact if $p_n < 2p_1$.

Let's look at Examples 2 and 3.

Example 2. For the basis $\{2047, 2048, 2049\}$ the number $X = 3758423681$ in RNS it is presented as $X \xrightarrow{RNS} (673, 1665, 353)$.

Example 3. For the basis $\{3, 5, 626604229\}$ the number $X = 3758423681$ in RNS it is presented as $X \xrightarrow{RNS} (2, 1, 625402536)$.

As can be seen from Example 3, if the compactness condition is not met, the third residue of the number in RNS has the same bit length as the number in the positional numeral system, which negates the advantage of RNS. The total computational delay depends on the largest modulo in the system.

4. Generating Compact RNS Bases

To generate compact sets of moduli we can use the method of constructing prime numbers which is used in the standard STB 1176.2-99 (and the Russian standard GOST R 34.10-94 which has ceased to function), which is based on Diemitko's theorem [21].

Theorem 1: Let $n = qR + 1$, where q is a prime odd number, R is an even number, $R < 4(q + 1)$, i.e., $n < (2q + 1)^2$. If $a < n$ is found:

1) $a^{n-1} \equiv 1(mod\ n)$,
2) $a^{\frac{n-1}{q}} \not\equiv 1(mod\ n)$, then n is a prime number.

Thus, if we have a prime number q, then, by searching even numbers R, we construct numbers $n = qR + 1$ and test them for primality according to Diemitko's theorem until we obtain a prime number. By the obtained number we can construct another prime number.

Algorithm 2 allows us to obtain a larger prime number p, having length $|p| = t$, starting from a smaller prime number q, whose length is $|q| = \left\lceil \frac{t}{2} \right\rceil$.

Algorithm 2: Generating prime numbers using Diemitko's theorem (PrimeNumbers).

Input: t – the required dimensionality of the prime number, q – a prime number
Output: p
1. $R = \left\lceil \frac{2^{t-1}}{q} \right\rceil + \left\lceil \frac{2^{t-1}\xi}{q} \right\rceil$
2. **if** $R \not\equiv 0(mod\ 2)$ **then**
2.1 $R = R + 1$
3. $u = 0$
4. $n = (R + u)q + 1$
5. **if** $n > 2t$ **then**
5.1 Return to step 1
6. **if** $2^{(n-1)} \equiv 1(mod\ n)$ **and** $2^{(R+u)} \not\equiv 1(mod\ n)$ **then**
6.1 $p = n$
6.2 **break**
7. **else**
7.1 $u = u + 2$
7.2 Return to step 4

The process uses a random variable ξ uniformly distributed on the interval $(0,1)$. This value is generated in a linear congruent manner. For each step of the algorithm, a new value of ξ is calculated.

Some prime numbers generated by this method may not be defined as such, because at step 6 the verification of the condition of Diemitko's theorem is carried out only for the number $a = 2$ and not for all $a < p$. However, the probability that a randomly chosen number a fulfils the conditions of Diemitko's theorem for a prime number n is $\left(1 - \frac{1}{q}\right)$. Using the check exclusively for $a = 2$ turns out to be quite sufficient to exclude only a small number of prime numbers from consideration. The advantage of choosing $a = 2$ is due to the fact that the degree expansion of the number 2 in the binary representation system is performed very efficiently.

Here is an example of generating a prime number using Diemitko's theorem.

Example 4. For $q = 3 = 11_2$, let us generate a prime number of length $t = 4$.

Let's find R at $\xi = 0.5$:

$$R = \left\lceil \frac{8}{3} \right\rceil + \left\lceil \frac{8 \cdot 0.5}{3} \right\rceil = 3.$$

To satisfy parity, $R = R + 1 = 4$. Candidate prime numbers $p = 4 \cdot 3 + 1 = 13$.

Since, $2^{12}(mod\ 13) \equiv 1$ and $2^4(mod\ 13) \not\equiv 1$.

Hence, the sought prime number $p = 13 = 1011_2$.

Thus, using Algorithm 2, Algorithm 3 is developed, which allows to generate compact bases with moduli of the form $p_i = Rq_i + 1$.

Algorithm 3: Generation compact bases.

Input: $\{q_1, q_2, \dots, q_n\}, t$

Output: $b = \{p_1, p_2, \dots, p_k\}$

1. b append PrimeNumbers(q_1, t)
2. **for** $i = 2, i < n, i++$ **do**

2.1 p =PrimeNumbers(q_1, t)

2.2 **if** $p < 2p_1$ **then**

2.2.1 b append p

In the next section, we will consider the performance of the proposed algorithm.

5. Performance Evaluation

Modelling and computational experiments were conducted on a computer equipped with a 2.80 GHz Intel Core i7-7700HQ processor, 8 GB of 1196 MHz DDR4 RAM and a 512 GB SSD, running Windows 10 Home, using the high-level programming language C++.

The first stage of the experiment involved comparing the speed of generating compact bases based on the Diemitko's theorem against methods for generating very large RNS bases. For comparison, two approaches were selected: the Pseudo-Mersenne number construction method and the general filtering method. The performance results of the algorithms are presented in Table 3.

Table 3. Comparison of execution time for different bases generation approaches, ms.

Number of modulo	General filtering	Construction base of Pseudo-Mersenne	Generating compact bases
8	10324	5328	4561
12	25211	9523	7531
16	50164	15433	13467
20	79057	19544	15389
32	165897	28413	23953

The compact bases generation method, based on the Diemitko's theorem, exhibits superior performance across all tested cases, with execution times substantially lower than both the Pseudo-Mersenne construction and general filtering approaches.

The experimental results demonstrate that the compact basis generation method is on average 15.5% faster than the Pseudo-Mersenne-based construction method and 75.7% faster than the general filtering approach when varying the number of moduli from 8 to 32. The maximum performance advantage is observed for the 32-moduli basis, where the compact method outperforms the Pseudo-Mersenne approach by 15.7% and the general filtering method by 85.6%. The second stage involved constructing sets of moduli with a dynamic range size from 32 to 128 bits and comparing them with a special set $\{2^n - 1, 2^n, 2^n + 1\}$. The comparison sets are presented in Tables 4 and 5. Next, the sets of moduli were compared in performing modular operations (Algorithm 1). The results of modular operations execution time are presented in Tables 6-8. Based on the presented data, we can conclude that on average compact bases provide the following speed gains for modular operations: by 11.87% for addition, by 12.08% for subtraction, and by 12.43% for multiplication. Thus, the use of compact bases allows speeding up calculations by about 12% on average for all operations compared to the use of moduli of a special set. Especially noticeable speed increase is observed for 96 and 128 bits, which indicates the prospect of using the algorithm of compact bases generation when increasing the digit capacity of numbers.

Table 4. Bases generated by algorithm 3 for modular operations modeling.

Dynamic range size, bits	**Bases**
32	$\{1823, 1997, 1997\}$
64	$\{521, 599, 613, 617, 647, 761\}$
96	$\{4217, 4447, 4951, 5279, 5281, 5461, 5521, 6521\}$
128	$\{16633, 17317, 17579, 17747, 20287, 20981, 21067, 22079, 24179\}$

Table 5. Moduli sets $\{2^n - 1, 2^n, 2^n + 1\}$ *for modular operations modeling.*

Dynamic range size, bits	**Bases**
32	$\{2047, 2048, 2049\}$
64	$\{4194303, 4194304, 4194305\}$
96	$\{4294967295, 4294967296, 4294967297\}$
128	$\{8796093022207, 8796093022208, 8796093022209\}$

Table 6. Results of number addition modeling in RNS, μs.

Dynamic range size, bits	32	64	96	128
Bases $\{2^n - 1, 2^n, 2^n + 1\}$	143.384	167.211	195.984	223.263
Bases generated by the algorithm 3	142.184	140.301	162.101	193.652

Table 7. Results of number subtraction in RNS, μs.

Dynamic range size, bits	32	64	96	128
Bases $\{2^n - 1, 2^n, 2^n + 1\}$	143.641	168.023	196.112	221.287
Bases generated by the algorithm 3	143.1	140.871	161.539	189.975

Table 8. Results of multiplication of numbers in RNS, μs.

Dynamic range size, bits	32	64	96	128
Bases $\{2^n - 1, 2^n, 2^n + 1\}$	144.544	172.382	198.073	224.632
Bases generated by the algorithm 3	143.624	141.593	163.122	194.162

6. Conclusion

This paper presents a method for generating compact RNS bases based on Diemitko's theorem. Experimental results reveal that the compact basis generation method reduces computation time by an average of 15.5% compared to Pseudo-Mersenne-based construction and 75.7% compared to general filtering. These advantages become more pronounced with increasing system size, reaching performance improvements of 15.7% and 85.6% respectively for 32-moduli bases. The method's efficiency extends to modular arithmetic operations, where it provides approximately 12% faster execution for addition, subtraction, and multiplication compared to traditional $\{2^n - 1, 2^n, 2^n + 1\}$ moduli sets.

Thus, the proposed RNS basis generation approach demonstrates high computational speed while enhancing the efficiency of modular arithmetic operations. This is particularly crucial for cryptographic applications that require processing of large numbers. Future research will focus on optimizing non-modular RNS operations using the moduli generated by the proposed algorithm.

References

[1]. Sousa L. Nonconventional computer arithmetic circuits, systems and applications. IEEE Circuits Syst. Mag, 2021, vol. 21, no. 1, pp. 6-40.

[2]. Garner H.L. The residue number system. Papers presented at the the March 3-5, 1959, western joint computer conference. ACM, 1959, pp. 146–153.

[3]. Луценко В.В., Бабенко М.Г., Хамидов М.М. Высокоскоростной метод перевода чисел из системы остаточных классов в позиционную систему счисления. Труды ИСП РАН, том 36, вып. 4, 2024 г., стр. 117-132. DOI: 10.15514/ISPRAS-2024-36(4)-9. / Lutsenko V.V., Babenko M.G., Khamidov M.M. High speed method of conversion numbers from residue number system to positional notation. Proceedings of the Institute for System Programming of the RAS, 2024, vol. 36, issue 4, pp. 117-132 (in Russian). DOI: 10.15514/ISPRAS-2024-36(4)-9.

[4]. Луценко В.В., Бабенко М.Г., Черных А.Н., Лапина М.А. Оптимизация алгоритма деления чисел в системе остаточных классов на основе функции ядра Акушского. Труды ИСП РАН, том 35, вып. 5, стр. 157-168. DOI: 10.15514/ISPRAS-2022-35(5)-11. / Lutsenko V.V., Babenko M.G., Tchernykh A.N., Lapina M.A. Optimization of a number division algorithm in the residue number system based on the Akushsky core function. Proceedings of the Institute for System Programming of the RAS, 2023, vol. 35, issue 5, pp. 157-168 (in Russian). DOI: 10.15514/ISPRAS-2022-35(5)-11.

[5]. Shiriaev E. Kucherov N., Babenko M., Nazarov A. Fast operation of determining the sign of a number in rns using the akushsky core function. Computation, 2023, vol. 11, no. 7, pp. 124.

[6]. Cardarilli G. C., Nannarelli A., Re M. RNS applications in digital signal processing. Embedded Systems Design with Special Arithmetic and Number Systems, 2017, pp. 181-215.

[7]. Schoinianakis D. Residue arithmetic systems in cryptography: a survey on modern security applications. Journal of Cryptographic Engineering, 2020, vol. 10, no. 3, pp. 249-267.

[8]. Nakahara H., Sasao T. A High-speed Low-power Deep Neural Network on an FPGA based on the Nested RNS: Applied to an Object Detector. 2018 IEEE international symposium on circuits and systems (ISCAS). – IEEE, 2018, pp. 1-5.

[9]. Ananda Mohan P. V. RNS in Cryptography. Residue Number Systems: Theory and Applications. – Cham: Springer International Publishing, 2016, pp. 263-347.

[10]. Fournaris A. P., Papachristodoulou L., Sklavos N. Secure and efficient rns software implementation for elliptic curve cryptography. 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). – IEEE, 2017, pp. 86-93.

[11]. Fournaris A. P., Papachristodoulou L., Batina L., Sklavos N. Secure and efficient RNS approach for elliptic curve cryptography, 2016.

[12]. Zalekian A., Esmaeildoust M., Kaabi A. Efficient implementation of NTRU cryptography using residue number system. International Journal of Computer Applications, 2015, vol. 124, no. 7.

[13]. Bajard J. C., Kaihara M., Plantard T. Selected RNS bases for modular multiplication. 2009 19th IEEE Symposium on Computer Arithmetic. – IEEE, 2009, pp. 25-32.

[14]. Bajard J. C., Fukushima K., Plantard T., Sipasseuth A. Generating very large RNS bases. IEEE Transactions on Emerging Topics in Computing, 2022, vol. 10, no. 3, pp. 1289-1301.

[15]. Lutsenko V., Zgonnikov M. Investigation of Neural Network Methods for Error Detection and Correction in the Residue Number System. International Workshop on Advanced Information Security Management and Applications. – Cham: Springer Nature Switzerland, 2024, pp. 194-206.
[16]. Omondi A. R., Premkumar A. B. Residue number systems: theory and implementation. – World Scientific, 2007, vol. 2.
[17]. Skavantzos A., Abdallah M., Stouraitis T. Large dynamic range RNS systems and their residue to binary converters. Journal of Circuits, Systems, and Computers, 2007, vol. 16, no. 02, pp. 267-286.
[18]. Molahosseini A. S., Teymouri F., Navi K. A new four-modulus RNS to binary converter. Proceedings of 2010 IEEE International Symposium on Circuits and Systems. – IEEE, 2010, pp. 4161-4164.
[19]. Lutsenko V.V., Kravtsov M.D., Gorlachev D.E., Mirny N.M. Research of special sets of moduli of the residue number system. Proceedings of the Institute for System Programming of the RAS, 2025, vol. 35, no. 5, pp. 157-168 (in Russian).
[20]. Kawamura S., Koike M., Sano F., Shimbo A. Cox-rower architecture for fast parallel montgomery multiplication. Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19. Springer Berlin Heidelberg, 2000. pp. 523-538.
[21]. Diemitko N. Generating multiprecision integer with guaranted primality. Proc. of the SIFIP Int. Conf. on Comp. Sci., IFIP Security 88, Amsterdam, 19-21 May, 1988. pp. 1-8.

Информация об авторах / Information about authors

Владислав Вячеславович ЛУЦЕНКО – аспирант кафедры вычислительной математики и кибернетики факультета математики и компьютерных наук имени профессора Н.И. Червякова ФГАОУ ВПО «Северо-Кавказский федеральный университет». Сфера научных интересов: высокопроизводительные вычисления, система остаточных классов, умный город, нейронные сети, интернет вещей.

Vladislav Vyacheslavovich LUTSENKO – postgraduate student, Department of Computational Mathematics and Cybernetics, Faculty of Mathematics and Computer Science named after Professor N.I. Chervyakov, North Caucasus Federal University. Research interests: high-performance computing, residue number system, smart city, neural networks, Internet of Things.

Михаил Григорьевич БАБЕНКО – доктор физико-математических наук, заведующий кафедры вычислительной математики и кибернетики факультета математики и компьютерных наук имени профессора Н.И. Червякова ФГАОУ ВПО «Северо-Кавказский федеральный университет». Сфера научных интересов: облачные вычисления, высокопроизводительные вычисления, система остаточных классов, нейронные сети, криптография.

Mikhail Grigoryevich BABENKO – Dr. Sci. (Phys.-Math.), Head of the Department of Computational Mathematics and Cybernetics, Faculty of Mathematics and Computer Science named after Professor N.I. Chervyakov, North Caucasus Federal University. His research interests include cloud computing, high-performance computing, residue number systems, neural networks, cryptography.

DOI: 10.15514/ISPRAS-2025-37(5)-4

Round-Trip Time Prediction Using Machine Learning Methods

[1,2] *I.A. Stepanov, ORCID: 0009-0003-1964-5001 <ivan_mipt@ispras.ru>*
[1] *R.E. Ponomarenko, ORCID: 0009-0009-5741-3627 <rerandom@ispras.ru>*
[1] *D.R. Golovash, ORCID: 0009-0006-5552-4428 <golovash@ispras.ru>*
[1] *A.Y. Pokidko, ORCID: 0009-0008-8981-8429 <a.pokidko@ispras.ru>*
[1,2,3,4] *A.I. Getman, ORCID: 0000-0002-6562-9008 <ever@ispras.ru>*

[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 (National Research University), 9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia.*

[3] *National Research University «Higher School of Economics», 20, Myasnitskaya ulitsa, Moscow, 101000, Russia.*

[4] *Lomonosov Moscow State University, 1, Leninskie Gory, Moscow, 119991, Russia.*

Abstract. The congestion control algorithms in the TCP protocol use RTT predictions indirectly or directly to determine congestion. The main algorithm for predicting RTT based on a weighted moving average is the Jacobson Algorithm. However, this algorithm may not work quite efficiently if the RTT is subject to a heavy-tailed distribution. In this paper, we propose an RTT prediction method based on supervised learning in both the offline and online cases. The results show improvement in the performance of algorithms based on supervised learning compared to the classical Jacobson algorithm in terms of MAPE, MAE, and MSE metrics. In addition, the high efficiency of online learning in comparison with offline learning in the case of data drift is shown.

Keywords: TCP; RTT prediction; online learning; Adaptive Random Forest regression.

For citation: Stepanov I.A., Ponomarenko R.E., Golovash D.R., Pokidko A.Y., Getman A.I. RTT prediction using offline and online learning. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 53-66. DOI: 10.15514/ISPRAS-2025-37(5)-4.

Предсказание времени приема-передачи с использованием методов машинного обучения

[1,2] *И.А. Степанов, ORCID: 0009-0003-1964-5001 <ivan_mipt@ispras.ru>*
[1] *Р.Е. Пономаренко, ORCID: 0009-0009-5741-3627 <rerandom@ispras.ru>*
[1] *Д.Р. Головаш, ORCID: 0009-0006-5552-4428 <golovash@ispras.ru>*
[1] *А.Ю. Покидько, ORCID: 0009-0008-8981-8429 <a.pokidko@ispras.ru>*
[1,2,3,4] *А.И. Гетьман, ORCID: 0000-0002-6562-9008 <ever@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*

[2] *Московский физико-технический институт, 141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9.*

[3] *Национальный исследовательский университет «Высшая школа экономики», 101978, Россия, г. Москва, ул. Мясницкая, д. 20.*

[4] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, г. Москва, Ленинские горы, д. 1.*

Аннотация. Время приема-передачи (RTT, Round-Trip Time) – время, которое требуется для отправки пакета от отправителя к получателю и возврата подтверждения, что пакет был получен. Алгоритмы управления перегрузками в протоколе TCP косвенно или напрямую используют предсказанные значения RTT для определения перегрузки сети. Основным алгоритмом для прогнозирования RTT на основе взвешенного скользящего среднего является алгоритм Джейкобсона. Однако этот алгоритм может работать не совсем эффективно, если RTT имеет распределение с тяжёлым хвостом, т.е. существуют редкие, но очень большие значения RTT. В этой статье мы предлагаем метод прогнозирования RTT, основанный на обучении с учителем, который может работать как в оффлайн режиме (с заранее собранной обучающей выборкой), так и в онлайн режиме (с поступлением данных в реальном времени и их последовательной обработкой). Полученные результаты показывают улучшение алгоритмов, основанных на машинном обучении, по сравнению с классическим алгоритмом Джейкобсона с точки зрения показателей MAPE, MAE и MSE. Кроме того, показана высокая эффективность онлайн обучения по сравнению с оффлайн обучением в случае дрейфа концепции или дрейфа данных.

Ключевые слова: транспортный протокол TCP; прогнозирование времени приема-передачи (RTT); онлайн-обучение; адаптивная регрессия случайного леса.

Для цитирования: Степанов И.А., Пономаренко Р.Е., Головаш Д.Р., Покидько А.Ю., Гетьман А.И. Предсказание RTT с использованием оффлайн и онлайн обучения. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 53–66 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)–4.

1. Introduction and Motivation

RTT (round-trip time) is the time required to send a data packet from the source to the recipient and back to the source. This is an important parameter in network performance. In addition, the retransmission timer (RTO) has an important role in the TCP protocol. This timer is set when sending a segment and its expiration serves as a congestion signal. The problem of choosing this timer is related to the fact that the RTT has a high variance from the point of view of a random variable, which significantly complicates the prediction of this value.

The prediction of RTT is an important component of congestion control algorithms (CCA). Packet loss-based CCAs such as TCP Reno and TCP Cubic indirectly use RTT information to determine congestion. In addition to loss-based CCA, there are CCAs that detect congestion directly from RTT: TCP Vegas, TCP Vegas-A. Therefore, for such methods, it is extremely important to accurately predict RTT one step ahead. Also, multipath technology has recently become very popular, allowing

the client to transfer data over multiple network paths. The scheduler, which determines the path to send the packet, makes decisions based on certain metrics, one of which is RTT. In this case, RTT prediction can also be very important.

They are usually based on the Jacobson algorithm, which predicts RTT using the moving average method. However, as some researchers have noted, the moving average method may not work well for values from distributions with a heavy tail, which may well include RTT. Therefore, a number of papers have been proposed that predict RTT using recurrent neural networks. Since recurrent neural networks require a large training dataset, its collection is an important component of the RTT prediction task. However, models of this class can often work inefficiently in terms of decision-making time, which can be critical in terms of congestion control.

In addition, due to the high variability of RTT, a model trained in one network environment (with a low RTT value) may be less effective in another network environment (with a high RTT value). This behaviour is due in part to data drift.

In order to avoid a drop in predictive ability during the transition from one environment to another, it makes sense to detect drift during model runtime, and in case of drift, online learning it based on new data.

Therefore, in this paper there is propose an online machine learning method with drift detection. The results obtained show an improvement in RTT prediction using this method compared to the Jacobson algorithm. At the same time, an improved prediction is observed in various network scenarios, in terms of the RTT value.

The rest of the article is structured as follows. Section II provides information on the structure of RTT and the classical methods of its measurement. Section III describes RTT prediction methods that use both probability distributions and machine learning. Section IV contains a statement of the problem of online learning and drift detection. Section V provides a description of our method. Section VI contains comparisons of the method implemented in this paper with the Jacobson algorithm.

2. Background

Using different concepts of RTT, it can be stated that:

$$RTT \approx delay_{propagation} + delay_{transmission} + delay_{queueing} + delay_{processing}$$

- $delay_{propagation}$ – the propagation delay is the time it takes for a signal to move from the sender to the receiver through physical media (such as cables or radio waves). It depends on the distance between the nodes and the speed of signal propagation in the environment.
- $delay_{transmission}$ – the transmission delay is the time required to transmit a data packet over a communication channel. The transmission delay depends on the packet size and bandwidth of the communication channel.
- $delay_{queueing}$ – the queue delay is the time during which a data packet is queued on the forwarding devices, waiting for the next packets to be transmitted.
- $delay_{processing}$ – the processing delay is the time required for packet processing on routers and end nodes. It includes the time required to process headers, check for errors, and perform other operations related to packet routing and processing.

It makes sense to consider RTT between sender and recipient as the sum of two main components: the constant component, which includes propagation delay and transmission delay, and the variable component, which includes queuing delay and processing delay. Queuing delay and processing delay are the main source of uncertainty in the prediction of RTT, as they depend on various components.

2.1 RTT measurement methods

To predict RTT using machine learning models, it is necessary to collect a dataset containing information about the RTT sequence. There are two ways to do this with ready-made tools.

The first is to use the ping command and send ICMP packets. However, ping does not always measure an accurate RTT. For example, when routers process ICMP packets during congestion, certain application flows may be prioritized. Thus, ICMP packets will generate RTTs that do not reflect the RTT that the priority traffic is encountering. In addition, some networks may block ICMP traffic, which also complicates the data collection process. The second way is to use the Wireshark tool: *tcp.analysis.rtt*. The third way to get an RTT value is by using TCP packet parameters such as Tsecr and Texp. However, in this case, the RTT accuracy will be limited to milliseconds.

2.2 The Jacobson algorithm

The first classical RTT prediction algorithm was the Jacobson algorithm, introduced in TCP Reno [1]. In this algorithm, the predicted RTT is subsequently used to calculate the RTO in the following form:

$$ERR = |(RTT_n - SRTT_{n-1})|$$
$$SRTT_n = \frac{7}{8}SRTT_{n-1} + \frac{1}{8}RTT_n$$
$$VAR_n = \frac{3}{4}VAR_{n-1} + \frac{1}{4}ERR$$
$$RTO = SRTT_n + 4VAR_n$$

Based on the moving average formula, we can see that:

$$SRTT_n = \frac{1}{8}RTT_n + \left(\frac{7}{8}\right)\cdot\frac{1}{8}RTT_{n-1} + \left(\frac{7}{8}\right)^2\cdot\frac{1}{8}RTT_{n-2} + \ldots$$

The usual estimate proposed by Jacobson works well in Gaussian distributed delay environments. However, as some researchers have noted, this algorithm may be inaccurate in environments with a different RTT distribution.

3. Related Work

There are two areas of work on RTT prediction: based on probability distributions and based on machine learning.

In several papers, RTT and, as a result, RTO are predicted based on the assumption that they are subject to a certain distribution. Thus, in [2], a method for approximate estimate of RTT was proposed based on the assumption that RTT is subject to the Weibull distribution. In [3], a method was proposed for a more detailed assessment of RTT based on the assumption that RTT is subject to a normal distribution. In [4], the authors proposed a method based on the calculation that the difference between neighboring values of RTT is subject to the Cauchy distribution. Using this assumption and Chebyshev's inequality, the authors can obtain the following estimate for the RTO:

$$RTO(k) = RTT(k-1) + \sqrt{\left(\frac{2\gamma\epsilon}{tan(\pi\phi)}\right) + \epsilon^2 - \gamma^2}$$

- γ– jitter dispersion
- ϕ– defined quality of service (QoS) parameter, which indicates the minimum fraction of time during which the prediction error is below the acceptable error ϵ.

These methods rely on assumptions about the distribution of RTT. However, the dynamic variability of RTT negatively affects the ability to accurately predict RTT in these methods, because the distribution of RTT can vary depending on the network environment.

A hybrid RTT prediction method based on geographical distance was proposed in [5]. The RTT prediction algorithm consisted of several stages. The first is an estimate of the distance between two IP addresses (sender and recipient). If the distance is less than 120 km, the RTT value was determined based on the database. If the distance is greater than 120 km, the RTT value was determined based on the trained model. The trained model was based on a decision tree that contained three features: Internet service provider, geographical distance between pairs of IP addresses and time of day. It is worth noting that distance is not always an informative feature, as it can change rapidly due to dynamic changes in network routes.

Recurrent neural networks have shown good predictive ability for predicting time series. As a result, a number of papers have appeared that predicted RTT based on previous values of RTT. The algorithm proposed in [6] has the following form:

$$ERR = |(RTT_n - SRTT_{n-1})|$$
$$SRTT_n = F(RTT_1, RTT_2 \dots RTT_{188})$$
$$VAR_n = \frac{3}{4} VAR_{n-1} + \frac{1}{4} ERR$$
$$RTO = SRTT_n + 4VAR_n$$

Here, F is a function implemented by a recurrent neural network. In [7], an RTT prediction method was proposed based on passive measurements collected at an intermediate node. The recurrent neural network (LSTM) was chosen as the prediction model. In [8], a lightweight version of the recurrent neural network GRU was proposed.

However, neural networks can require high computational costs, which is critical in the context of RTT prediction. Therefore, it makes sense to consider classical machine learning models (Random Forest, Linear regression).

It is worth noting that the RTT prediction study in the above papers was given only for the offline case. However, the efficiency of the algorithm in the offline and online case may vary greatly. Therefore, both offline and online scenarios will be considered in this paper.

4. Online learning

The task of online learning can be formulated as follows. Let's give a sequence of features and target values $(x_i, y_i)_{i=1}^n$. $a(x, w)$ - parametric model, $L(w, y)$ – loss function. At each step i, the following set of actions is performed:

- getting object features x_i
- the prediction is made based on the received object $a(x_i, w_{i-1})$
- getting y_i
- calculation of the loss function $L\big(y_i, a(x_i, w_{i-1})\big)$
- updating the weights of the model based on the loss function w_i

It is worth noting that incremental learning, unlike online learning, works with batches, while online learning uses only one object at each step. Otherwise, the two approaches are very similar in the context of the task under consideration.

4.1 Drift detection

Data drift is a phenomenon in which the statistical properties of the data used to train a machine learning model change over time. This means that the distribution of the input data in the real world

no longer corresponds to the distribution of the data on which the model was trained. Ultimately, due to this phenomenon, the accuracy of the model degrades.

More strictly, let there be some target variable and a set of features defining this variable. The drift is understood as a change in the distribution of the input data $P(X)$, the target variable $P(Y)$, or the relationship between them $P(Y|X)$ over time. It is worth noting that there are several types of drift detection in research.

Input data drift: let's give the initial distribution of input data (features) $P_0(X)$ and some distribution of data $P_t(X)$ at time t. It is said that there is a drift in the input data if:

$$P_0(X) \neq P_t(X)$$

Drift of the target variable (Label Drift): let's give the initial distribution of the target variable $P_0(Y)$and some target variable $P_t(Y)$ at time t. To say that there is a drift in the label data if:

$$P_0(Y) \neq P_t(Y)$$

Concept Drift: let's give the initial dependence distribution $P_0(Y|X)$ and $P_t(Y|X)$ at time t. To say that there is a concept drift in the data if:

$$P_0(Y|X) \neq P_t(Y|X)$$

There are a large number of ways to detect drift. These include statistical methods: the Kolmogorov—Smirnov test [9], the Chi-square test [10], the Darling-Anderson test [11], methods based on autoencoders [12], as well as methods based on the ARIMA model [13].

4.2 ADWIN

The ADWIN (Adaptive Windowing) [14] algorithm is a method that solves the problem of detecting changes in statistical characteristics of data, such as mean or variance. ADWIN uses the hypothesis of equality of the averages between different parts of the data window. If these hypotheses are rejected, it means that data drift has occurred.

The algorithm divides the window Winto two sub-parts: $W_0$and W_1. Then, for each part, the following are calculated: n_0, n_1- size of window $W_0$and $W_1\mu_0, \mu_1$- average values $W_0$and W_1. If the difference between the observed mean values $|(\mu_0 - \mu_1)|$exceeds ϵ_{cut}, the algorithm considers that the distributions in $W_0$and $W_1$are different, and deletes the old part $W_0$of the window. In this case, $\epsilon_{cut}$is calculated as follows:

$$m = \frac{n_0 \cdot n_1}{n_0 + n_1}$$

$$\delta' = \frac{\delta}{n}$$

$$\epsilon_{cut} = \sqrt{\left(\frac{1}{2m}\right) \cdot ln\left(\frac{4}{\delta'}\right)}$$

4.3 Adaptive Random Forest regressor

Adaptive Random Forest (ARF) [15-16] is an online learning algorithm that adapts to concept drift. The main idea of the algorithm is to have an ADWIN-based drift detector for each tree of a Random Forest. If the detector detects a change, the corresponding one is removed and retrained on the new dataset. Thus, the ensemble of trees adapts to the new distribution.

4.4 Online learning and drift detection

The general scheme of online learning used in this work is shown in Fig. 1.

The online learning process consists of several important parts: the main dataset, an Adaptive Random Forest, and a drift detector based on the ADWIN method. In the process of online learning, new objects are received at the input of the algorithm. The drift detector checks for drift between new objects and the main dataset, which is constantly being updated. If drift is detected, the Adaptive Random Forest is updated based on new data; if not, the Adaptive Random Forest remains unchanged. Thus, the model's stability to changing environmental conditions is achieved.

In this case, the ADWIN algorithm determines the drift for the normalized value: $|(y_{true} - y_{predict})|$. Thus, if the distribution of $|(y_{true} - y_{predict})|$changes significantly, the ADWIN algorithm detects the drift.

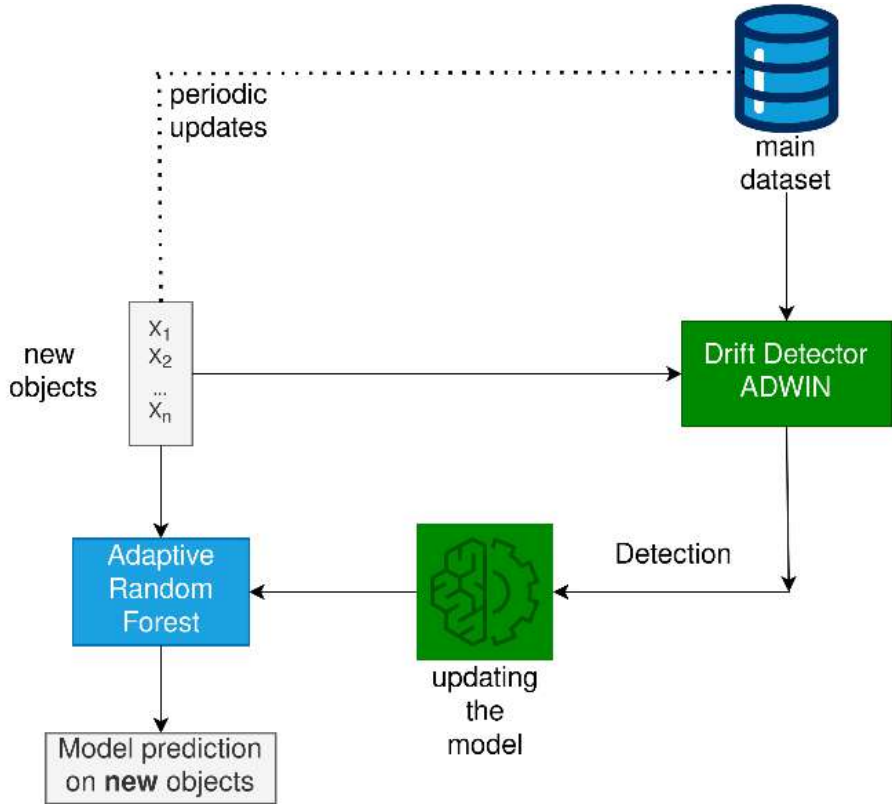


Fig. 1. Online learning and drift detection scheme.

5. Implementation

5.1 Problem formulation

From the point of view of supervised learning, the RTT prediction task is a regression task.

$$f: X \to Y$$

- X– features object
- Y- predicted RTT value

It makes sense to consider the following characteristics of a TCP stream as features.

RTT: In most studies, it is proposed to use sequential RTT values as features. This paper also examines these values for RTT prediction.

TTL: As noted earlier, the geographical distance between two hosts can change dynamically and is not always an informative feature in the RTT prediction task. However, it makes sense to use the TTL parameter, which is the IPv4 field of the packet header that specifies the maximum number of routers (hop count) through which the packet can pass before it is dropped. Each time a packet passes through the router, the TTL value decreases by one. Therefore, this parameter can be used as features for RTT prediction. In IPv6, the Hop Limit field is an analogue of the TTL parameter from IPv4. From the point of view of the problem under consideration, the Hop Limit and TTL are equivalent parameters.

Bytes in Flight: This value indicates how much data (in bytes) have been sent from the sender, but have not yet been confirmed by the recipient. The congestion control algorithm strives to maximize the use of the transmission channel so that the number of bytes in flight is approximately equal to BDP (Bandwidth-delay Product). Therefore, it can be stated that there is some connection between RTT and the number of bytes in the flight and use this feature in the task under consideration.

Thus, the following features vector is used for prediction RTT_n:

$$RTT_{n-1}\dots RTT_{n-k}, bytes_{n-1}\dots bytes_{n-k}, TTL_{n-1}\dots TTL_{n-k}$$

In this formula, k is a parameter that indicates the number of previous values used for prediction. The search for the optimal value of k, which preserves the high performance of the algorithm, will be described later in the paper.

5.2 Machine learning models

In this paper, the following machine learning algorithms were investigated within the framework of the problem under consideration: Linear regression with regularization, Decision Tree and Random Forest, as well as recurrent neural networks.

After examining the collected dataset, a high linear relationship was found between the predicted RTT and the features discussed above. This behavior motivates the use of linear regression with *L1* and *L2* regularization in the context of the task.

Random Forest is an algorithm that has proven itself well in working with data containing a large number of noises. In terms of RTT prediction, an abnormal value of this value caused by some external factors can be considered noise. Therefore, it makes sense to consider this model in the context of the task under consideration.

Recurrent neural networks have proven themselves well in the context of sequence prediction. Classical RNNs can have gradient attenuation problems when the network needs to remember information from the distant past: RTT prediction based on a large number of previous values. LSTM has a better ability to store information over long time intervals, but this model has a complex structure and may require high computing resources. The GRU model is simpler in terms of structure. In this paper, all three models for solving the RTT prediction problem will be investigated for a detailed analysis.

5.3 Training data

In the training process, the following dataset was collected, simulating 3 situations: a user is uploading files, an online game, and a regular web interaction.

For a wide variety of data and, consequently, for more efficient prediction of the model in a variety of network scenarios, a dataset was collected in a wide range of network characteristics: bandwidth from 10 Mbps to 100 Mbps, distance between sender and receiver from 100 km to 1200 km, as well as the use of both IPv4 and IPv6 protocols.

The characteristics of the collected dataset are shown in Table 1.

Table 1. Training Data.

Scenario	Number of objects	μ RTT, ms	σ^2 RTT, ms^2
Uploading files	922063	60.37	1344
Uploading files	1791633	9.89	144
Uploading files	1731034	40.75	58
Online game	30012	31.85	738
Web interaction	38250	10.10	140

- μ – average of RTT value
- σ^2 – variance of RTT value

It should be noted that the number of objects in the case of loading is significantly higher than in the other two scenarios. This is due to the fact that the complexity of obtaining objects in an online game scenario and in a web interaction scenario is much more complicated than in an upload scenario. However, section VI explores this problem for both balanced dataset and unbalanced dataset (number of objects in the loading scenario).

As noted above, a dataset with a true RTT value is needed to train the model. Experiments have shown that using the TCP packet option to measure accurate RTT does not provide significant advantages over *tcp.analysis.rtt* in the context of the task under consideration. Therefore, in this work, a *tcp.analysis.rtt* was used to obtain the correct RTT values.

6. Evaluation

This section presents the main results of the implemented algorithms in both offline and online scenarios. In addition, a comparison of machine learning algorithms with the classical Jacobson algorithm is presented.

6.1 Offline scenario balanced dataset

In this subsection, the considered models are trained on a balanced dataset containing objects from the upload scenario, the online game scenario, and the web interaction scenario. The models are tested on a dataset that also contains an equal proportion of objects in all three scenarios. The results obtained are presented in Table 2.

From the results obtained, it can be stated that in this scenario, a Random Forest shows the best result in terms of all metrics. It can also be noted that neural networks do not provide significant improvements compared to simpler algorithms in the context of the problem under consideration. The best value of k is understood as the smallest k, with an increase in which the error decreases slightly.

6.2 Offline scenario unbalanced dataset

In this subsection, the considered models are trained on an unbalanced dataset containing objects only from the loading scenario. The models are tested on a dataset that contains an equal proportion of objects in all three scenarios. The results obtained are presented in Table 3.

The results show that from the point of view of RTT prediction, the scenarios of different network situations do not differ.

Table 2. Offline scenario (balanced dataset).

Algorithm	best k	MSE	MAE	R^2	MAPE
ElasticNet	7	45.65	2.10	0.94	8.43
Random Forest	**4**	**40.36**	**1.80**	**0.94**	**6.75**
RNN	14	48.33	2.21	0.92	12.91
LSTM	14	47.98	2.13	0.93	11.29
GRU	14	48.50	2.20	0.92	12.58
The Jacobson algorithm	-	59.50	2.45	-	22.31

Table 3. Offline scenario (unbalanced dataset).

Algorithm	best k	MSE	MAE	R^2	MAPE
ElasticNet	8	43.37	2.00	0.94	7.97
Random Forest	**4**	**38.32**	**1.70**	**0.95**	**6.40**
RNN	11	48.24	2.19	0.92	12.84
LSTM	11	47.84	2.11	0.93	11.17
GRU	11	48.03	2.17	0.92	12.44
The Jacobson algorithm	-	59.30	2.67	-	23.24

From the results obtained, it can be argued that in this scenario, a Random Forest shows the best result in terms of all metrics. It can also be noted that neural networks do not provide significant improvements compared to simpler algorithms in the context of the problem under consideration. The best value of *k* is understood to be the smallest value of *k*, with an increase in which the error decreases slightly.

6.3 Online scenario

Due to the high dynamism of network environments, machine learning models trained on one traffic may not work well enough in traffic with other characteristics.

In this subsection, an online learning method based on an Adaptive Regression Forest with drift detection using the ADWIN method is proposed. The code implementing this training uses the River library [17].

The first dataset was collected in a low RTT network environment $objects: 60000, \mu = 8.67ms, \sigma = 1.24ms^2$, while the second dataset was collected in an environment with high RTT $objects: 20000, \mu = 73.93ms, \sigma = 2759.20ms^2$ and RTT have distribution with a heavy tail (Fig. 2).

A Random Forest was trained based on 50,000 objects in the first dataset, and then Random Forest tested on 10,000 objects of the first dataset and 20,000 objects of the second dataset. Thus, the case was considered when a model trained on a dataset with certain network characteristics was tested on a dataset with other network characteristics.

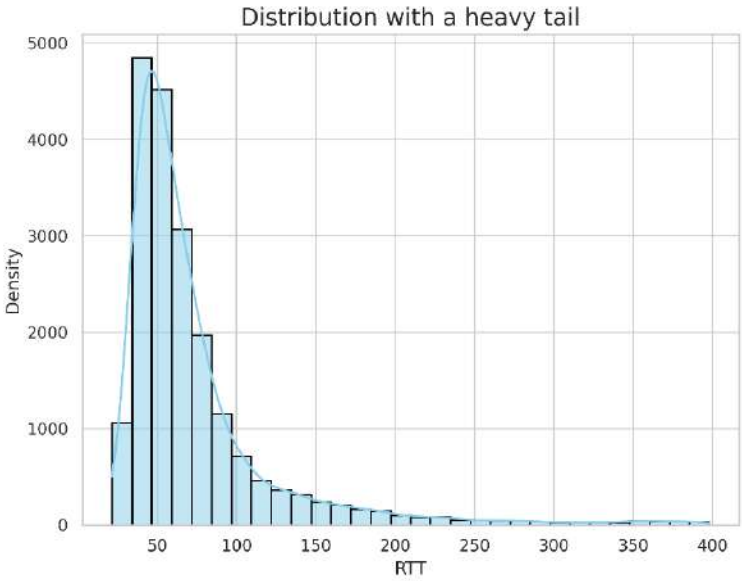


Fig. 2. Distribution with a heavy tail.

In the second experiment, the Adaptive Random Forest was trained on the 50000 objects of first dataset. On the second dataset, the algorithm was trained online using drift detection. This detection was used to more accurately train a Random Forest, in which the forest trees that solved the problem were most poorly replaced by new trees. The results obtained are shown in Fig. 3. The results obtained show the effectiveness of online learning in this task: the overall value of the MAPE metric does not deteriorating as critically as in the case of offline learning.

In the second pair of experiments, the dataset with a higher RTT $objects: 60000, \mu = 68.69ms, \sigma^2 = 2039.15ms^2$ value was the first, while the dataset with a lower RTT $objects: 20000, \mu = 8.70ms, \sigma = 1.22ms^2$ value was the second. The results obtained are shown in Fig. 4.

In this case, online learning also shows improvement. At the same time, offline learning shows very bad results.

7. Conclusions

In this paper, algorithms for RTT prediction using offline and online learning were presented. The results show that the algorithms based on learning works better in terms of MAPE, MSE, and MAE metrics in network environments with a wide range of network characteristics than the classic Jacobson algorithm.

However, as shown in this article, offline learning can be ineffective in dynamically changing network environments. To solve this problem, an online learning method with drift detection was proposed. The results show that in the case of online learning, the prediction efficiency does not deteriorate or deteriorates slightly when the environment changes.

The results obtained allow us to identify the following areas of future work:

- integration of online learning algorithms implemented in this paper into classical congestion control algorithms (TCP Reno, TCP CUBIC),
- study of the performance of classical congestion control algorithms using RTT prediction using online learning.

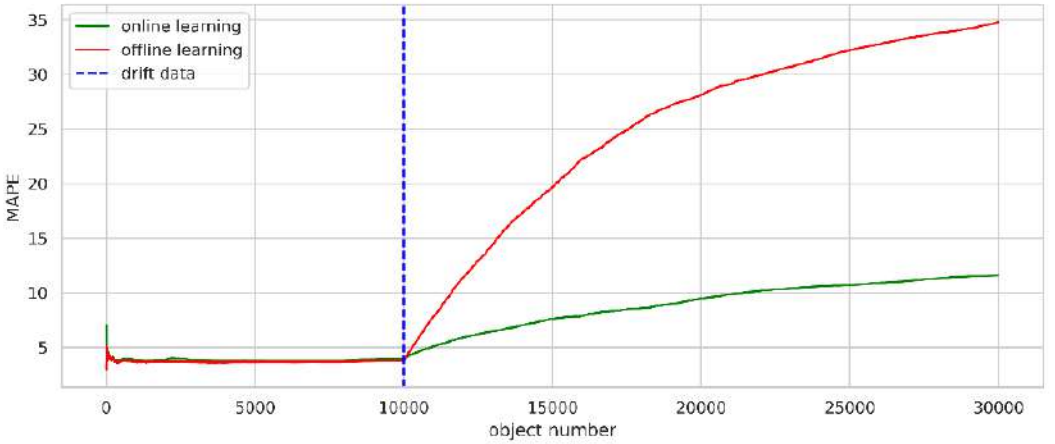


Fig. 3. Offline and Online learning (the first pair of experiments).

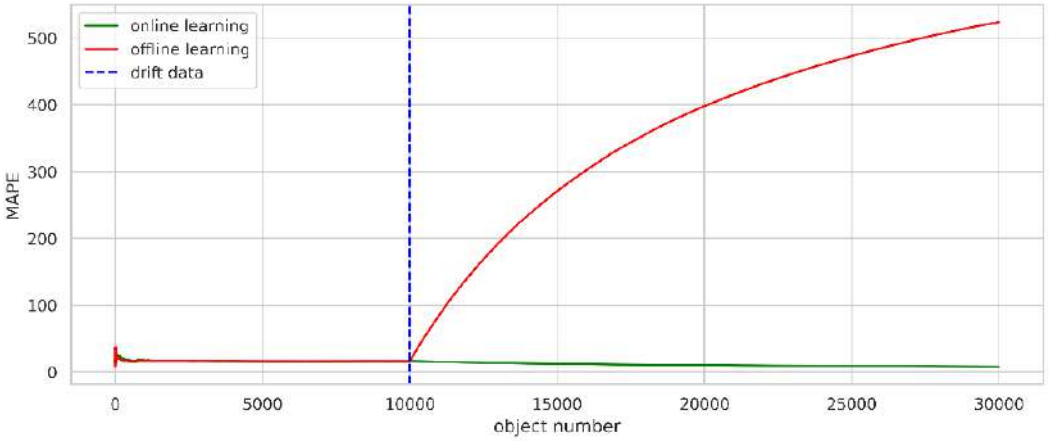


Fig. 4. Offline and Online learning (the second pair of experiments).

Список литературы / References

[1]. Mo, J., La, R. J., Anantharam, V., and Walrand, J. (1999, March). Analysis and comparison of TCP Reno and Vegas. In IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320) (Vol. 3, pp. 1556-1563). IEEE.

[2]. Hernández, José-Alberto, and Iain W. Phillips. "Weibull mixture model to characterise end-to-end Internet delay at coarse time-scales." IEE Proceedings-Communications 153.2 (2006): 295-304.

[3]. Cerroni, W., Foschini, L., Grabarnik, G. Y., Shwartz, L., and Tortonesi, M. (2017). Estimating delay times between cloud datacenters: A pragmatic modeling approach. IEEE Communications Letters, 22(3), 526-529.

[4]. Rizo-Dominguez, L., Munoz-Rodriguez, D., Vargas-Rosales, C., Torres-Roman, D., and Ramirez-Pacheco, J. (2014). RTT prediction in heavy tailed networks. IEEE Communications Letters, 18(4), 700-703.

[5]. Hu, Wen, Zhi Wang, and Lifeng Sun. "Guyot: a hybrid learning-and model-based RTT predictive approach." 2015 IEEE International Conference on Communications (ICC). IEEE, 2015.

[6]. Dasgupta, B., D. Valles, and S. McClellan. "Estimating TCP RTT with LSTM Neural Networks." Proceedings on the International Conference on Artificial Intelligence (ICAI). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2019.

[7]. Hagos, D. H., Engelstad, P. E., Yazid, A., and Griwodz, C. (2019, October). A deep learning approach to dynamic passive RTT prediction model for TCP. In 2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC) (pp. 1-10). IEEE.
[8]. Dong, Ai, Zhijiang Du, and Zhiyuan Yan. "Round trip time prediction using recurrent neural networks with minimal gated unit." IEEE Communications Letters 23.4 (2019): 584-587.
[9]. Berger, Vance W., and YanYan Zhou. "Kolmogorov–smirnov test: Overview." Wiley statsref: Statistics reference online (2014).
[10]. Berkson, Joseph. "Some difficulties of interpretation encountered in the application of the chi-square test." Journal of the American Statistical Association 33.203 (1938): 526-536.
[11]. Engmann, Sonja, and Denis Cousineau. "Comparing distributions: the two-sample anderson-darling test as an alternative to the kolmogorov-smirnoff test." Journal of applied quantitative methods 6.3 (2011).
[12]. Jaworski, Maciej, Leszek Rutkowski, and Plamen Angelov. "Concept drift detection using autoencoders in data streams processing." International Conference on Artificial Intelligence and Soft Computing. Cham: Springer International Publishing, 2020.
[13]. Nau, Robert. "The mathematical structure of ARIMA models." Duke University Online Article 1.1 (2014): 1-8.
[14]. Bifet, Albert, and Ricard Gavalda. "Learning from time-changing data with adaptive windowing." Proceedings of the 2007 SIAM international conference on data mining. Society for Industrial and Applied Mathematics, 2007.
[15]. Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfharinger, B., ... and Abdessalem, T. (2017). Adaptive random forests for evolving data stream classification. Machine Learning, 106, 1469-1495.
[16]. Gomes, H. M., Barddal, J. P., Ferreira, L. E. B., and Bifet, A. (2018, April). Adaptive random forests for data stream regression. In ESANN.
[17]. Montiel, J., Halford, M., Mastelini, S. M., Bolmier, G., Sourty, R., Vaysse, R., ... and Bifet, A. (2021). River: machine learning for streaming data in python. Journal of Machine Learning Research, 22(110), 1-8.

Информация об авторах / Information about authors

Иван Александрович СТЕПАНОВ – аспирант, стажёр-исследователь ИСП РАН, ассистент кафедры информатики и вычислительной математики МФТИ. Сфера научных интересов: анализ сетевого трафика с помощью машинного обучения.

Ivan Alexandrovich STEPANOV – postgraduate student of the ISP RAS, intern researcher at ISP RAS, an assistant at the Department of Computer Science and Computational Mathematics at MIPT. Research interests: network traffic analysis using machine learning.

Роман Евгеньевич ПОНОМАРЕНКО – младший научный сотрудник ИСП РАН. Научные интересы: архитектура программного обеспечения, оптимизация программ, глубокий анализ сетевого трафика.

Roman Evgenevich PONOMARENKO – junior researcher at ISP RAS. Research interests: software architecture, program optimization, deep packet inspection.

Денис Ростиславович ГОЛОВАШ — лаборант ИСП РАН, студент ВМК МГУ. Сфера научных интересов: анализ сетевого трафика с помощью машинного обучения.

Denis Rostislavovich GOLOVASH is a laboratory assistant at the ISP RAS, a student at the Moscow State University. Research interests: network traffic analysis using machine learning.

Антон Юрьевич ПОКИДЬКО – стажер-исследователь отдела компиляторных технологий ИСП РАН. Научные интересы: дрейф в машинном обучении и нейронных сетях, трансферное обучение, федеративное обучение, онлайн обучение, анализ сетевого трафика.

Anton Yurevich POKIDKO – research intern at Compiler Technology department of ISP RAS. Research interests: drift in machine learning and neural networks, transfer learning, federated learning, online learning, network traffic analysis.

Александр Игоревич ГЕТЬМАН – кандидат физико-математических наук, старший научный сотрудник ИСП РАН, ассистент ВМК МГУ и МФТИ, доцент ВШЭ. Сфера научных интересов: анализ бинарного кода, восстановление форматов данных, анализ и классификация сетевого трафика.

DOI: 10.15514/ISPRAS-2025-37(5)-5

Coloring Symbolic Memory Graphs to Detect DRM-Specific Errors in Linux Drivers

E.M. Orlova, ORCID: 0009-0003-1654-3085 <e.orlova@ispras.ru>
A.A. Vasilyev, ORCID: 0000-0002-5738-9171 <vasilyev@ispras.ru>
O.M. Petrov, ORCID: 0009-0004-6245-9615 <o.petrov@ispras.ru>

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

Abstract. This paper discusses a particular type of subtle use-after-free errors in the Direct Rendering Manager (DRM) subsystem of the Linux kernel. These errors occur due to incorrectly allocated memory for structures accessible from user space via device callbacks. To detect these errors, we use a shape analysis based on the Symbolic Memory Graph (SMG) domain. We introduce the coloring of allocated memory to track its origin. Among 186 Linux DRM drivers, we have found 6 violations of the proposed rule.

Keywords: Linux drivers; use-after-free; shape analysis; software model checking; symbolic memory graphs.

For citation: Orlova E.M., Vasilyev A.A., Petrov O.M. Coloring Symbolic Memory Graphs to Detect DRM-Specific Errors in Linux Drivers. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025. pp. 67-80. DOI: 10.15514/ISPRAS-2025-37(5)-5.

Acknowledgements. The authors would like to thank Vadim Mutilin, colleagues from the Linux Verification Center, and the maintainers of the Linux DRM subsystem for their feedback and comments.

Окрашивание символьных графов памяти для выявления ошибок, специфичных для DRM-драйверов Linux

Е.М. Орлова, ORCID: 0009-0003-1654-3085 <e.orlova@ispras.ru>
А.А. Васильев, ORCID: 0000-0002-5738-9171 <vasilyev@ispras.ru>
О.М. Петров, ORCID: 0009-0004-6245-9615 <o.petrov@ispras.ru>

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

Аннотация. В статье рассматривается одна трудновоспроизводимая ошибка типа use-after-free в подсистеме Direct Rendering Manager (DRM) ядра операционной системы Linux. Её причиной является некорректный способ выделения памяти, доступной для пользовательского кода через обратные вызовы устройства. Для поиска ошибок работы с памятью мы используем анализ на основе символьных графов памяти (SMG). Чтобы отследить способ выделения памяти, мы добавили ей цвет. Среди 186 проанализированных драйверов DRM ОС Linux было найдено 6 нарушений предложенного правила.

Ключевые слова: драйверы Linux; уязвимость use-after-free; анализ динамической памяти; автоматическая статическая верификация; символьные графы памяти.

Для цитирования: Орлова Е.М., Васильев А.А., Петров О.М. Окрашивание символьных графов памяти для выявления ошибок, специфичных для DRM-драйверов Linux. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 67-80 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)-5

Благодарности: Авторы выражают благодарность Вадиму Мутилину, коллегам из Технологического центра исследования безопасности ядра Linux, а также разработчикам, поддерживающим подсистему DRM в Linux, за их отзывы и комментарии.

1. Introduction

The Linux operating system kernel is a widely used software system consisting of 25 million lines of code. Put simply, it consists of the kernel core and various subsystems and device drivers. To use a graphics processing unit (GPU), a user program invokes a system call so that the kernel core dispatches the appropriate callback in the corresponding device driver in the Direct Rendering Manager (DRM) kernel subsystem.

This paper discusses a particular type of errors related to the incorrect use of device resource management (devres) in the DRM subsystem. Incorrect memory allocation of structures accessible from user space can lead to a use-after-free memory access. Such outcomes can be detected with dynamic analysis, but situations in which the target errors cause the kernel to crash are specific and quite rare.

On the other hand, static verification methods [1] aim at detecting such subtle errors. Klever [2-3] is a software verification platform capable of automated static verification of industrial software systems using software model checking tools such as CPAchecker [4]. As many Linux errors are in its device drivers [5], Klever is tailored for bug-finding in the Linux subsystems.

Klever decomposes the kernel source code into modules, provides the environment models based on typical device usage, runs the verification tool, and displays the results, e.g. visualizes the reported error traces. Using this method, several hundred bugs in Linux subsystems were found and reported [6].

To verify memory safety, CPAchecker uses a shape analysis based on the Symbolic Memory Graph domain [7-10]. The analysis represents a program memory state as a bipartite graph, with its nodes being memory objects (concrete regions and abstracted linked lists) and symbolic values.

Contribution. We manually analyzed the Linux DRM subsystem and found a documented recommendation [11] on correct allocation that had been overlooked in 13 files [12]. This can lead

to potential use-after-free errors, some of which have already been reported by Kernel Address Sanitizer (KASAN).

We formulated a more general rule in natural language and formally specified it using Klever. To verify this rule, we adapted the memory analysis in the CPAchecker verification tool by adding color to the symbolic memory graphs. We evaluated this approach on 186 DRM modules, with all 6 reported violations of the rule manually confirmed. The results are discussed in comparison with the violations we found using Coccinelle [13]. The corresponding Coccinelle rule (*semantic patch*) was submitted to the kernel [14]. Finally, we are working on fixes for the discovered errors, and one patch has already been accepted upstream [15].

2. Problem Statement

There is a static driver structure through which device instances are managed. Through a certain interface, a device instance is accessible from the user space. Even if the driver is disabled, the device instance will still exist as long as it has at least one user. If memory is allocated incorrectly, the DRM device structure (or structures used by it) is automatically freed when the driver is unbound. Thus, while the device instance still exists, a user can cause access to this freed structure.

2.1 DRM device instance

At the core of every DRM driver is a `drm_driver` structure. It contains static information that describes the driver and features it supports, and pointers to methods that implement the DRM API. This structure is also used to create a device instance, which is then initialized and registered, providing callbacks accessible from the user space.

A device instance for the DRM driver is represented by the `drm_device` structure. It is allocated and initialized with `devm_drm_dev_alloc()` (or deprecated `drm_dev_alloc()`). After initialization of all the various DRM device subsystems when everything is ready for user space, the device instance can be published using `drm_dev_register()` [11].

When cleaning up, everything is done in reverse. First, the device instance is unpublished with `drm_dev_unregister()`. Then any other resources allocated at device initialization are cleaned up and drop the driver's reference to `drm_device` using `drm_dev_put()`. It is important to note that if `drm_device` still has some resource handles open when the driver is unbounded, the release of `drm_device` instance does not happen immediately, but only after the last handle is closed. Before that, `drm_device` remains user-accessible. This is why any allocation or resource which is visible to user space must be released only when the final `drm_dev_put()` is called, and not when the driver is unbound from the underlying physical `struct device`. Otherwise, using the device may result in accessing freed memory.

This imposes a restriction on which functions can be used to allocate structures that are accessible from user space through a `drm_device` instance.

2.2 Some Linux kernel memory allocation functions

Let's look at some of the memory allocation functions in more depth.

- `kmalloc()` – a kernel-space function similar to user-space `malloc()`. The memory allocated with this function must be freed by calling the `kfree()` function.
- `devm_kmalloc()` – devres-managed `kmalloc()`. Memory allocated with this function is automatically freed on driver detach. Its lifetime is linked to the device structure, a pointer to which is passed as a parameter.
- `drmm_kmalloc()` – DRM-managed `kmalloc()`. Memory allocated with this function is automatically freed on the final `drm_dev_put()`. Its lifetime is linked to the `drm_device` structure, a pointer to which is passed as a parameter.

`drmm_kmalloc()` is recommended to allocate the memory for the aforementioned structures accessible from user space. Then the memory will be automatically released when the `drm_device` is destroyed and only after its registration is canceled, as there will be no risk of accessing the released memory. The correct work of a DRM driver is shown in Fig. 1.

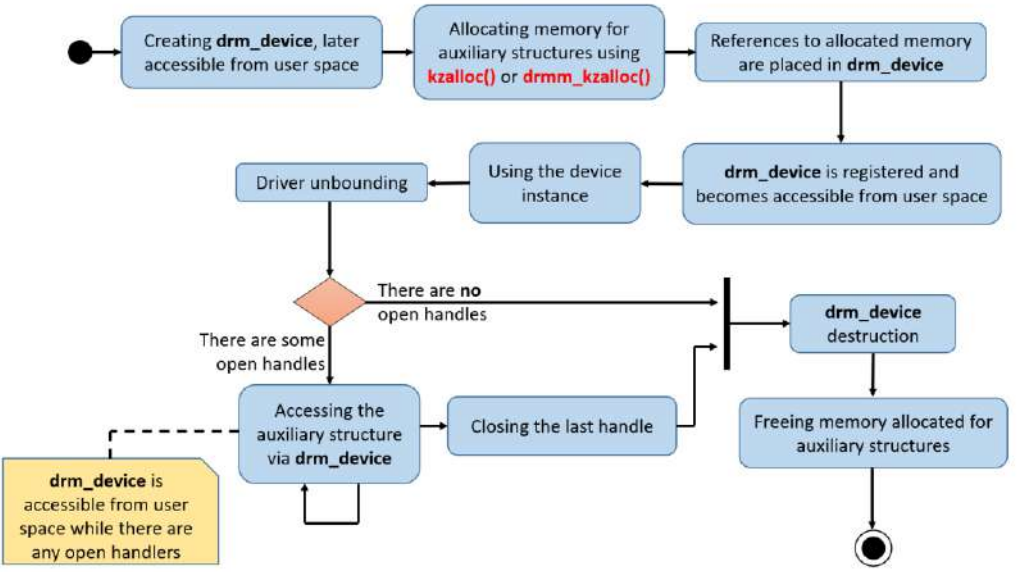


Fig. 1. A correct operation of a DRM driver.

Contrarily, using `devm_kmalloc()` in most of these cases is a mistake, since the release may occur ahead of time. The driver will still work, but sudden crashes will happen periodically. The reason is shown in Fig. 2. If the device still has a user after the driver detach, the user can try to access the structure previously allocated with `devm_kmalloc()` and released on driver detach. There are vulnerabilities of this type in the Linux kernel. Therefore, structures accessible from the user space after `drm_device` registration should not be allocated with devres-managed functions like `devm_kmalloc()`.

Accordingly, there is a documented restriction on the second argument of 5 functions that initialize preallocated DRM-specific structures – `drm_encoder_init`, `drm_connector_init`, `drm_connector_init_with_ddc`, `drm_crtc_init_with_planes`, and `drm_universal_plane_init` – namely, the second argument should not be allocated with `devm_kzalloc` or similar devres functions. This is a more obvious violation of the general rule.

3. Related Work

There is a wide variety of approaches and tools for bug finding in industrial software systems. Here, we limit the discussion to those most relevant to the Linux kernel [16].

3.1 Dynamic analysis

Dynamic analysis tools [17] typically look for a class of issues occurring in the running kernel. One example is Kernel Address Sanitizer (KASAN) [18] which can detect invalid memory accesses such as out-of-bounds and use-after-free errors.

The presence of the target errors in the code poses a risk of accessing freed memory, so they can be detected by KASAN. Indeed, it is mentioned in comments of some target error fixes accepted into the kernel and can be used to confirm the fix.

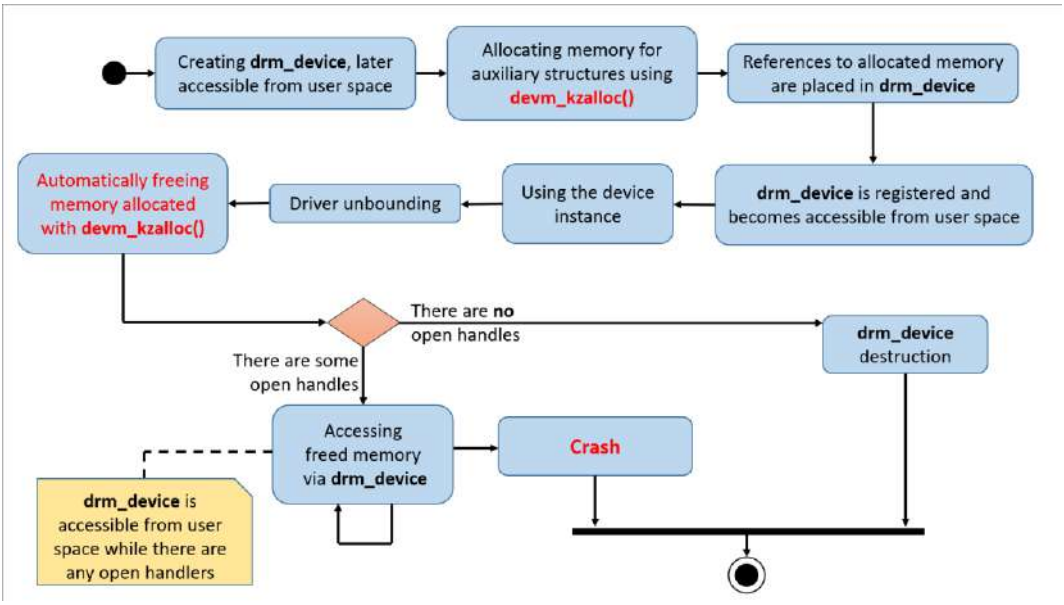


Fig. 2. An operation of a DRM driver with a target error.

However, dynamic analysis can only check the parts of the code that are reachable through the tests. Moreover, the target errors can cause races in very specific situations. Another drawback is the need for particular hardware to run certain drivers. These limitations effectively prevent dynamic analysis from reliably detecting the target errors.

3.2 Static analysis

Lightweight static analysis, such as abstract syntactic tree (AST) analysis and data-flow analysis (DFA), searches for defects in the program source code without execution. It can detect potential errors, vulnerabilities and non-compliance with standards at early stages of development, thereby saving time and resources. The tools most closely tied to the Linux kernel and used by its development community include Coccinelle, Sparse, and Smatch [16].

Coccinelle [13] is a program matching and transformation tool focused on patterns in source code structure. It operates on semantic patches – high-level patterns that resemble git patches but are abstracted with metavariables and ellipses. The tool employs a temporal logic (extended CTL [19]) to reason about a function's control flow. The strength of the tool is the relative ease of writing semantic patches for known patterns and its ability to generate patches automatically. However, its major limitation for our purposes is the lack of any data-flow reasoning and the very limited support for interprocedural analysis.

Sparse [20] is a source parser and analyzer that extends the C type system with kernel-specific annotations. These include address-space qualifiers to prevent mixing user and kernel pointers, and endianness markers to detect incorrect byte-order handling. Sparse also performs simple intraprocedural DFA for context-tracking [21] (i.e. matching context counters on entry and exit against annotations) and uses this together with locking annotations to warn about imbalanced or missing lock acquisition and release. Applying type-checking to the target errors would require manually annotating pointers along the data flow, which is more effort than manual inspection of the DRM drivers.

More conventional static analyzers, such as Smatch and Svace, use full-fledged data-flow analysis. Smatch [22] builds on Sparse by adding a "cross-function flow analysis" [23]. It is path-sensitive and can track a range or multiple values for a given variable. Svace [24], used by the Linux

Verification Center (LVC) [25, 26], employs both syntactic and flow analysis. It leverages a bottom-up technique: functions lower in the call graph are summarized using data-flow analysis and symbolic execution, and these summaries are then reused at call sites when analyzing functions higher in the call graph.

Although Smatch and Svace are effective at detecting various generic and kernel-specific errors, lightweight static analysis is generally unable to find the target errors in the complex scenarios characteristic of our target errors. The inherent trade-off between scalability and false positive rate forces the use of heuristics, which sacrifices soundness and precision.

3.3 Software model checking

Software model checking, or static verification, can be considered a heavyweight form of static analysis. The approach aims at thorough exploration of a program's state space, which allows detecting subtle errors, e.g. data races, and thus is used for critical system verification [1]. The drawbacks of the approach are high resource consumption and the frequent need for handwritten formal specifications.

Klever [2, 3] is a verification platform designed to automate the software model checking for industrial systems. For scalability, Klever decomposes large codebases into smaller, verifiable modules. Specifications needed for particular requirements or missing function bodies can be written in C [27]. The environment model, i.e. calls to the module, is provided based on the typical scenarios of device usage [28-29].

Klever together with the CPAchecker static verification tool have been used to find several hundred errors in the kernel, including memory safety violations [7], data races [30], and errors specific to Linux device drivers [3, 5]. To find the target errors, we need both write the specification for our DRM-specific rule and modify the underlying memory analysis to remember the allocating function for allocated memory regions.

4. Colored Symbolic Memory Graphs

To verify memory safety, CPAchecker [4] uses a shape analysis based on the Symbolic Memory Graph (SMG) domain [7-8] that first appeared in the Predator shape analyzer [9-10, 31].

The analysis represents a program memory state as a labeled bipartite graph of memory objects and symbolic values, and edges between them. A "has-value" edge from a memory object to a symbolic value means that the value is stored in the object (the offset and bitsize are labeled on the edge). A "points-to" edge from a symbolic value to a memory object means that the value points to the object (again, the offset from the start of the object is labeled on the edge).

To distinguish objects allocated in a certain way, we have introduced memory coloring for the analysis. The color of an allocated region is determined by the allocating function:

- *DRM* for the `drm_device` structures allocated by `drm_dev_alloc()` or `devm_drm_dev_alloc()`,
- *DEVM* for the `devm_kmalloc()`-allocated memory,
- and default (colorless) memory is allocated by all other functions.

Now, we can reformulate our rule in terms of the colored memory graphs: *DRM*-colored memory objects should not store pointers to *DEVM*-colored memory objects.

Simplified erroneous code is provided in Listing 1. A probe driver method allocates its own device structure and `drm_device` structure. When the analysis traverses the first line with a call to `devm_kzalloc()`, it adds a new heap object (shown as "DEVM" in Fig. 3) and a new symbolic value ("s1") that points to its start. As the pointer to the new allocation is stored in the variable `ldev`, a has-value edge `ldev` → s1 is added, too. As `devm_kzalloc()` is a colored function, the allocation gets the corresponding color (shown as red).

```
// allocate DRM device ddev with the given dev as parent
struct drm_device *ddev = drm_dev_alloc(&drv_driver, &pdev->dev);
// allocate the specific device with actually the same dev as parent
struct ltdc_device *ldev =
  devm_kzalloc(ddev->dev, sizeof(*ldev), GFP_KERNEL);
// the rule violation
ddev->dev_private = (void *)ldev;
// ldev may be accessed after it is released before ddev is released
```

Listing 1. Simplified erroneous code (before the patch)
from `drivers/gpu/drm/stm/drv.c`*, functions* `stm_drm_platform_probe` *and* `drv_load`*.*

The `ddev` initialization is analyzed in the same manner, with the new allocation colored DRM (shown as green). The result SMG (without the labels on the edges) can be seen in Fig. 3 on the left; unimportant parts (such as previous stack frames and global variables) are not shown. After the assignment in the last line, the SMG looks like in Fig. 3 on the right. Note that *DRM*-colored allocation now has a field (DRM → s1) that points to a *DEVM*-colored allocation (s1 → DEVM). When such an assignment happens, the analysis reports an error.

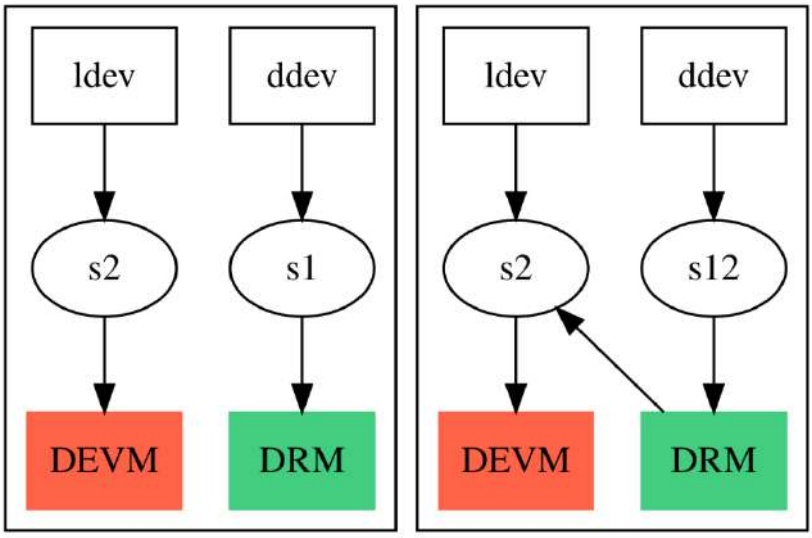


Fig. 3. Left: the symbolic memory graph for Listing 1 before assignment.
Right: the symbolic memory graph after assignment;
the presence of DRM → s2 → DEVM path is a violation of the proposed rule.

5. Specification of DRM subsystem in Klever

Klever decomposes the kernel into modules, with the result that CPAchecker runs on each of the modules separately [29]. This solves the issue of running heavy analysis on large code, but there is a problem with functions defined in other modules. Their bodies are not visible to CPAchecker, so during the verification of a module, it assumes that such a function is pure, i.e. it does not affect the analyzed code. If a function is important for finding the target error (e.g. it initializes a pointer important for the analysis), one has to write a *model* for it that CPAchecker will traverse instead of the original function.

Klever allows us to write such models for functions and replaces every call to the original function in the module with a call to the given model [28]. This is implemented using aspect-oriented programming [27]. Suppose there is a function `foo()` in the kernel code that we want to replace

with a model. Then we write a `ldv_foo()` function in the `.c` file, and we specify in the `.aspect` file that instead of `foo()` calls, `ldv_foo()` should be actually called.

The capability to replace a function with a model is also useful if it needs to be abstracted from insignificant details (simplified) or given a new feature, such as color for memory it allocates.

In our case, some memory allocation and releasing should be colored appropriately. To do this, a special "color" function was called in the bodies of the models (`ldv_color_drm_kmalloc()` or `ldv_color_devm_kmalloc()`, depending on the desired color). Models were also required for a number of imported functions in which bindings between structures were created. Basically, instead of a function initializing all fields of the structure, a model filled in several pointers, the value of which influenced the success of the target error search.

For the DRM subsystem, we have modelled the following functions in Klever:

- Devres specification. It includes `devm_kmalloc()` and its analogs (`devm_kzalloc()`, `devm_kcalloc()`, `devm_kmalloc_array`). Memory allocated with this function is automatically freed on driver detach. These functions paint memory in the color *DEVM*. If a reference to memory of the color *DEVM* appears in `drm_device`, an error is reported.
- `drm_device` managed resources specification. It includes memory allocation functions (`drmm_kmalloc()` and its analogs, which paint memory in the color *DRM*) and implementation of various ways to free it.
- Special functions used to allocate and deallocate `drm_device` and `drm_driver` memory and to initialize them: `drm_dev_alloc()`, `drm_dev_init()`, `drm_dev_release()`, etc. The function models responsible for initialization create references needed to find target errors.
- Models of functions that initialize structures used by DRM device. `drm_encoder_init()` for `srtuct drm_encoder`, `drm_universal_plane_init()` for `srtuct drm_plane`, and so on. In them, the structures are linked to `drm_device`, and if their memory was allocated incorrectly – i.e., with the *DEVM* color – an error is detected at the moment of storing a reference to such memory.
- Models of other functions that create and destroy links between structures: `drm_dev_put()`, `get_device()`/`put_device()`, `kref_init()`/`kref_put()`.

6. Evaluation

We applied our approach to 186 loadable DRM driver modules from `drivers/gpu/drm/` in Linux 5.10.238, targeting the ARM architecture with the `allmodconfig` build configuration.

The experiment was carried out with Klever, derived from version 4.0.1 [32], together with our fork of CPAchecker [33] on a machine with an Intel Core i7-11700 2.50GHz CPU (8 cores, 16 threads), 2x16 GB DDR4 RAM, and an SSD.

In total, verification has taken 10 h of CPU time (40 min of wall time). We have limited the CPAchecker verification tool to 270 s per module; it has used up to 1.3 GB for a module and consumed 4.2 h of CPU time in total. Table 1 details the results of the verification.

- 108 modules were verified as safe, with CPAchecker exhausting all reachable states without detecting any target or generic memory safety error.
- 33 modules resulted in verifier timeout, where CPAchecker did not complete within the allotted time.
- For 3 modules, CPAchecker stopped after encountering a recursive call.
- 17 driver modules were not verified due to a composition problem, where Klever was unable to compose a module to verify due to atypical module init or exit, or missing declarations.

The analysis reported 25 modules as unsafe:

- 6 target errors;
- 1 generic memory error, specifically a non-target use-after-free;
- 18 false alarms for generic memory errors. These were primarily due to analysis imprecision (e.g., inability to calculate a dereferenced address). One of them was caused by inline assembler code in the sources.

Table 1. Verification results for the 186 Linux 5.10.238 DRM drivers.

Verdict	Count	%
Unsafe	**25**	**13**
target error	6	3.2
use-after-free	1	0.5
false alarm	18	9.7
Safe	**108**	**58**
Unknown	**53**	**28**
verifier timeout	33	17.7
recursion in module	3	1.6
composition problem	17	9.1
Total:	***186***	***100***

6.1 Estimating Missed Errors with Coccinelle

We used Coccinelle to estimate the amount of the target errors in the kernel code and to assess the false negative rate of our approach. As the presence of an error-prone pattern implies the need to fix multiple files in a module, we count the reported modules instead of matches for Coccinelle.

Following the documented restriction, we wrote the **arg** rule illustrated in Listing 2. It finds a devres-managed memory pointer passed as the second argument to one of the 5 drm-init functions with the documented restriction discussed in Section 2. We used Coccinelle to find 5 more functions – `drm_writeback_connector_init`, `drm_crtc_init`, `drm_plane_init` `drm_bridge_connector_init`, and `drm_simple_encoder_init` – that are simple wrappers to those and should thus have the same restriction applied. While we can continue to elaborate the rule, in practice we do not expect much more alarms.[1]

```
devm =@p devm_kzalloc(...);
...
drm_crtc_init_with_planes(@q e,<+...devm...+>,...)
```

*Listing 2. A snippet of the **arg** rule for DEVM-allocated second argument to a `drm_*_init` function.*

All target errors identified by Klever involved an assignment of DEVM-allocated memory pointer to the `dev_private` field of a DRM-allocated `drm_structure`. Moreover, 3 modules reported by Klever were not reported by the **arg** rule. This motivated a second Coccinelle rule, **field** (Listing 3) designed to detect assignments of a DEVM-allocated pointer to a field of a DRM-allocated structure. See more elaborated rules as submitted to the kernel in [14].

[1] There are a considerable number of cases where a DEVM-allocated pointer is first assigned to another local variable which is later passed to a drm-init function. However, these occur in modules already reported by the simpler rule.

```
drm = drm_dev_alloc(...);
...
devm = devm_kzalloc(...);
...
drm->f =@p <+...devm...+>;
```

*Listing 3. A snippet of the **field** rule for assigning a DEVM-allocated pointer to a DRM-allocated **field**.*

Table 2 presents a per-module comparison between the findings from Klever and the two Coccinelle rules, 27 modules in total. The column Klever shows the outcomes of our verification runs, while Coccinelle/field and Coccinelle/arg mark the modules in which the corresponding rule found violations. All 27 modules are reported by one of the Coccinelle rules; notably, only 4 are reported by both. Although our analysis targeted Linux 5.10.238, many of the bugs are still present in recent versions (6.17). In the Klever column, the outcomes are encoded as follows:

- target error – analysis reported a violation of the color rule (true positive);
- non-target alarm – a reported generic memory error turned out to be a false alarm;
- safe – full state-space exploration without detecting violations of the color rule or generic memory safety;
- unknown (timeout) – CPAchecker exceeded the allocated CPU time;
- unknown (recursion) – CPAchecker stopped on encountering a recursive function call;
- unknown (oom) – out-of-memory during module composition;
- unknown (comp. iss.) – compilation issue during module composition;
- unknown (arch) – module not included in the ARM build.

6.2 Error classification

True Positives. As shown in Table 2, Klever successfully identified target errors in 6 modules. All 6 modules were also reported by at least one Coccinelle rule: 5 modules were reported by the **field** rule and 3 by the **arg** rule. Notably, the assignment to the field in `stm/stm-drm` module was not reported by the **field** rule because one of the allocations happens in another function, and handling such interprocedural cases is limited in Coccinelle.

True Negatives. We did not assess true negatives.

False Positives. We found no false positives among the 6 target errors reported by Klever.

False Negatives. Klever missed a violation in 21 modules reported by Coccinelle: 8 reported by the **field** rule and 15 modules reported by the **arg** rule, respectively. Of these, 11 misses can be attributed to the limitations of our approach (timeouts, recursion, oom, comp. arch, non-target alarms). For the 10 modules reported as safe, the coverage appears to be lacking, as the relevant functions were not reached. This suggests the need to refine or add the specifications for the modules so the analysis can reach the DRM functions.

7. Conclusion

We have discussed a subtle use-after-free error in Linux DRM drivers that originates from misusing managed memory allocation for device structures. To find such errors, we proposed a coloring rule, introduced such coloring to the SMG analysis in the CPAchecker verification tool, and wrote specifications for the respective functions of the DRM subsystem.

For the specification and component-wise verification of 186 modules in the DRM subsystem, we have used the Klever verification platform. Klever was able to carry out the verification for 169 modules and reported 25 of them as unsafe. Among these, 6 modules contain a target error, and 1 module contains a generic memory error (use-after-free).

Moreover, we developed two Coccinelle rules: **arg**, which finds violations of the documented restriction, and **field**, which was motivated by the pattern in the errors found by Klever. While **arg**

reports errors in 18 modules, **field** reports 9 additional modules. Together, these approaches provide complementary coverage and demonstrate the effectiveness of combining lightweight and heavyweight methods.

Future work. We plan to continue submitting patches for the discovered errors. We also intend to refine and extend the specifications to improve the coverage across DRM modules.

Table 2. Comparison of target errors found by Klever and Coccinelle.

DRM module	Klever	Coccinelle	
arc/arcpgu	non-target alarm		arg
arm/hdlcd	safe	field	arg
arm/mali-dp	unknown (timeout)	field	
atmel-hlcdc/atmel-hlcdc-dc	non-target alarm		arg
fsl-dcu/fsl_dcu_drm	**target error**	**field**	
ingenic/ingenic-drm	non-target alarm		arg
lima/lima	**target error**	**field**	
meson/meson-drm	unknown (recursion)	field	arg
meson/meson_dw_hdmi	safe		arg
msm/msm	unknown (oom)		arg
panfrost/panfrost	unknown (arch)	field	
pl111/pl111_drm	safe	field	
rcar-du/rcar_du	non-target alarm	field	
rockchip/rockchip_drm	unknown (timeout)	field	
shmobile/shmob_drm	**target error**	**field**	**arg**
sti/sti-drm	safe		arg
stm/stm-drm	**target error**		**arg**
sun4i/sun4i-drm	unknown (timeout)	field	
5 modules: sun4i/sun4i-{backend,drm-hdmi,tcon,tv} and sun4i/sun8i-mixer	5 safe		5 arg
tilcdc/tilcdc	**target error**	**field**	**arg**
tve200/tve200	**target error**	**field**	
vc4/vc4	unknown (timeout)		arg
zte/zx_drm	safe		arg
Modules with target errors:	**6 (22%)**	**13 (48%)**	**18 (67%)**

References

[1]. E.M. Clarke, T.A. Henzinger, H. Veith, and R. Bloem. Handbook of Model Checking. Springer International Publishing, Cham. 2018. DOI: 10.1007/978-3-319-10575-8.

[2]. I.S. Zakharov, M.U. Mandrykin, V.S. Mutilin, E.M. Novikov, A.K. Petrenko, and A.V. Khoroshilov. Configurable toolset for static verification of operating systems kernel modules. Programming and Computer Software, vol. 41, no. 1. 01.01.2015. pp. 49–64. DOI: 10.1134/S0361768815010065.

[3]. I. Zakharov, E. Novikov, and I. Shchepetkov. Klever: Verification Framework for Critical Industrial C Programs. 2023. DOI: 10.48550/arXiv.2309.16427.

[4]. D. Baier, D. Beyer, P.-C. Chien, M.-C. Jakobs, M. Jankola, M. Kettl, N.-Z. Lee, T. Lemberger, M. Lingsch-Rosenfeld, H. Wachowitz, and P. Wendler. Software Verification with CPAchecker 3.0: Tutorial and User Guide. Formal Methods. 2025. pp. 543–570. DOI: 10.1007/978-3-031-71177-0_30.

[5]. V.S. Mutilin, E.M. Novikov, and A.V. Khoroshilov. Analysis of typical faults in Linux operating system drivers. Trudy ISP RAN/Proc. ISP RAS, 2012, vol. 22, pp. 349–374 (in Russian). DOI: 10.15514/ispras-2012-22-19.

[6]. Found Bugs by Klever. [Online]. Available at: https://github.com/ldv-klever/klever?tab=readme-ov-file#found-bugs, accessed 09.09.2025.

[7]. A.A. Vasilyev. Static verification for memory safety of Linux kernel drivers. Trudy ISP RAN/Proc. ISP RAS, 2018, vol. 30, issue 6, pp. 143–160. DOI: 10.15514/ISPRAS-2018-30(6)-8.

[8]. A.A. Vasilyev and V.S. Mutilin. Predicate Extension of Symbolic Memory Graphs for the Analysis of Memory Safety Correctness. Programming and Computer Software, vol. 46, no. 8, 01.12.2020, pp. 747–754. DOI: 10.1134/S0361768820080071.

[9]. K. Dudka, P. Peringer, and T. Vojnar. Byte-Precise Verification of Low-Level List Manipulation. in F. Logozzo and M. Fähndrich (eds). Static Analysis. Springer Berlin Heidelberg, Berlin, Heidelberg. 2013. pp. 215–237. DOI: 10.1007/978-3-642-38856-9_13.

[10]. K. Dudka, P. Muller, P. Peringer, V. Šoková, and T. Vojnar. Algorithmic Details behind the Predator Shape Analyser. 2024. DOI: 10.48550/arXiv.2403.18491.

[11]. DRM Internals – The Linux Kernel documentation. [Online]. Available at: https://www.kernel.org/doc/html/latest/gpu/drm-internals.html, accessed 29.09.2025.

[12]. E. Orlova. [PATCH v4] drm/stm: Avoid use-after-free issues with crtc and plane. [Online]. Available at: https://lore.kernel.org/all/20240216125040.8968-1-e.orlova@ispras.ru/, accessed 06.10.2025.

[13]. J.L. Lawall and G. Muller. Coccinelle: 10 Years of Automated Evolution in the Linux Kernel. USENIX Annual Technical Conference. 2018. [Online]. Available at: https://www.usenix.org/system/files/conference/atc18/atc18-lawall.pdf, accessed 06.10.2025.

[14]. O. Petrov. [PATCH] cocci: drm: report devm-allocated arguments and fields. [Online]. Available at: https://lore.kernel.org/all/20250924140126.23027-1-o.petrov@ispras.ru/, accessed 24.09.2025.

[15]. E. Orlova. drm/stm: Avoid use-after-free issues with crtc and plane. [Online]. Available at: https://web.git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=19dd9780b7ac673be95bf6fd6892a184c9db611f, accessed 15.07.2024.

[16]. M. Schmitt. Linux kernel device driver testing. How are device drivers being tested? Master's Thesis, Institute of Mathematics and Statistics, University of São Paulo, São Paulo. 17.10.2022. DOI: 10.11606/D.45.2022.tde-30112022-152524.

[17]. A. Konovalov. Sanitizing the Linux kernel: On KASAN and other Dynamic Bug-finding Tools. Linux Security Summit Europe. 2022. [Online]. Available at: https://www.youtube.com/watch?v=KmFVPyHyfqQ.

[18]. K. Serebryany, D. Bruening, A. Potapenko, and D. Vyukov. AddressSanitizer: A Fast Address Sanity Checker. USENIX ATC 2012. 2012. [Online]. Available at: https://www.usenix.org/conference/usenixfederatedconferencesweek/addresssanitizer-fast-address-sanity-checker, accessed 06.10.2025.

[19]. J.L. Lawall, J. Brunel, N. Palix, R.R. Hansen, H. Stuart, and G. Muller. WYSIWIB: A declarative approach to finding API protocols and bugs in Linux code. DSN'09 – The 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. 2009. pp. 43–52. DOI: 10.1109/DSN.2009.5270354.

[20]. N. Brown. Sparse: a look under the hood. 2016. [Online]. Available at: https://lwn.net/Articles/689907/, accessed 06.10.2025.

[21]. L. Torvalds. Sparse 'context' checking. [Online]. Available at: https://lwn.net/Articles/109066/, accessed 18.09.2025.

[22]. N. Brown. Smatch: pluggable static analysis for C. 22.06.2016. [Online]. Available at: https://lwn.net/Articles/691882/, accessed 06.10.2025.

[23]. D. Alden. Finding locking bugs with Smatch. 11.06.2025. Write-up of Dan Carpenter's talk at Linaro Connect 2025. [Online]. Available at: https://lwn.net/Articles/1023646/, accessed 06.10.2025.

[24]. A. Belevantsev, A. Borodin, I. Dudina, V. Ignatiev, A. Izbyshev, S. Polyakov, E. Velesevich, and D. Zhurikhin. Design and Development of Svace Static Analyzers. 2018 Ivannikov Memorial Workshop (IVMEM). 2018. pp. 3–9. DOI: 10.1109/IVMEM.2018.00008.

[25]. Linux Verification Center — Static Analysis (in Russian). [Online]. Available at: https://portal.linuxtesting.ru/activity.html#menu3, accessed 29.09.2025.

[26]. Found by Linux Verification Center (linuxtesting.org) with SVACE. [Online]. Available at: https://web.git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/log/?qt=grep&q=Found+by+Linux+Verification+Center+(linuxtesting.org)+with+SVACE, accessed 29.09.2025.

[27]. E.M. Novikov. An approach to implementation of aspect-oriented programming for C. Programming and Computer Software, vol. 39, no. 4. 07.2013. pp. 194–206.
[28]. I.S. Zakharov, V.S. Mutilin, and A.V. Khoroshilov. Pattern-based environment modeling for static verification of Linux kernel modules. Programming and Computer Software, vol. 41, no. 3. 05.2015. pp. 183–195. DOI: 10.1134/S036176881503007X.
[29]. I. Zakharov and E. Novikov. Compositional Environment Modelling for Verification of GNU C Programs. 2018 Ivannikov ISPRAS Open Conference. 2018. pp. 39–44. DOI: 10.1109/ISPRAS.2018.00013.
[30]. P.S. Andrianov. Analysis of Correct Synchronization of Operating System Components. Programming and Computer Software, vol. 46, no. 8. 01.12.2020. pp. 712–730. DOI: 10.1134/S0361768820080022.
[31]. Predator. [Online]. Available at: https://www.fit.vut.cz/research/group/verifit/public/tools/predator/, accessed 29.09.2025.
[32]. Klever 4.0.1. [Online]. Available at: https://github.com/ldv-klever/klever/tree/v4.0.1/, accessed 18.03.2025.
[33]. CPAchecker 702bc1a. [Online]. Available at: https://github.com/ldv-klever/cpachecker/commit/702bc1a36f663d0e1bac13e6c6752e61828e6ac8, accessed 21.03.2025.

Информация об авторах / Information about authors

Екатерина Михайловна ОРЛОВА – студентка магистратуры факультета вычислительной математики и кибернетики МГУ, лаборант Института системного программирования РАН. Сфера научных интересов: статический анализ и верификация ядра Linux.

Ekaterina Mikhaylovna ORLOVA – Master's student at the Faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University (MSU), lab assistant at the Institute for System Programming of the RAS. Research interests: static analysis and verification of the Linux kernel.

Антон Александрович ВАСИЛЬЕВ – младший научный сотрудник Института системного программирования им. В.П. Иванникова РАН. Сфера научных интересов: статическая верификация и анализ программ.

Anton Aleksandrovich VASILYEV – junior researcher at the Ivannikov Institute for System Programming of the RAS. Research interests: static verification, software model checking, static program analysis.

Олег Максимович ПЕТРОВ – аспирант и стажёр-исследователь Института системного программирования им. В.П. Иванникова РАН. Сфера научных интересов: статическая верификация и анализ исходного кода программ, delta debugging.

Oleg Maximovich PETROV – postgraduate student and intern researcher at the Ivannikov Institute for System Programming of the RAS. His research interests include software model checking, static program analysis, delta debugging.

DOI: 10.15514/ISPRAS-2025-37(5)-6

Detection of SQL Injection Attacks through the Network Logs Using Machine Learning Methods

[1] *M.A. Lapina, ORCID: 0000-0001-8117-9142 <mlapina@ncfu.ru>*
[1] *N.R. Kapshuk, ORCID: 0009-0004-3644-7530 <kapshuknik06@gmail.com>*
[2] *M.A. Rusanov, ORCID: 0009-0000-7069-7542 <mix.rusanoff@yandex.ru>*
[1] *E.F. Timofeeva, ORCID: 0000-0001-5824-4778 <teflena@mail.ru>*

[1] *Faculty of Mathematics and Computer Science named after Professor N.I. Chervyakov, North Caucasus Federal University, 1, Pushkina str., Stavropol, 355017, Russia.*

[2] *Institute of Information Technology, Moscow University of Finance and Law, building 1, 17, Serpukhovskiy val str., Moscow, 115191, Russia.*

Abstract: The article examines machine learning methods for detecting the introduction of SQL code into the network logs using the KNIME program, based on finding patterns between incoming features and subsequent forecasting in a binary classification problem. Unlike existing works, this article examines the effectiveness of five tree-based machine learning methods. The content and sequence of work stages are presented. The highest results were shown by the Random Forest method (accuracy – 97.58%; area under the ROC curve is 0.976).

Keywords: machine learning; KNIME; classification; dataset; data selection; SQL injection; threat detection on the network; detection of suspicious patterns; protection of web applications.

For citation: Lapina M.A., Kapshuk N.R., Rusanov M.A., Timofeeva E.F. Detection of SQL injection attacks through the network logs using machine learning methods. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 81-92. DOI: 10.15514/ISPRAS-2025-37(5)-6.

Обнаружение атак с использованием SQL-инъекций по сетевым журналам с помощью методов машинного обучения

[1] *М.А. Лапина, ORCID: 0000-0001-8117-9142 <mlapina@ncfu.ru>*
[1] *Н.Р. Капшук, ORCID: 0009-0004-3644-7530 <kapshuknik06@gmail.com>*
[2] *М.А. Русанов, ORCID: 0009-0000-7069-7542 <mix.rusanoff@yandex.ru>*
[1] *Е.Ф. Тимофеева, ORCID: 0000-0001-5824-4778 <teflena@mail.ru>*

[1] *Факультет математики и компьютерных наук имени профессора Н.И. Червякова,*
Северо-Кавказский федеральный университет,
Россия, 355017, г. Ставрополь, ул. Пушкина, д.1.

[2] *Институт информационных технологий,*
Московский финансово-юридический университет,
Россия, 115191, г. Москва, ул. Серпуховский вал, д. 17, корп. 1.

Аннотация: В статье рассматриваются методы машинного обучения для обнаружения внедрения SQL-кода в сетевые журналы с помощью программы KNIME, основанные на поиске закономерностей между входящими признаками и последующем прогнозировании в задаче бинарной классификации. В отличие от существующих работ, в этой статье рассматривается эффективность пяти методов машинного обучения на основе деревьев. Представлено содержание и последовательность этапов работы. Наиболее высокие результаты показал метод "Случайный лес": точность – 97,58%; площадь под кривой ошибок (ROC-кривой) – 0,976.

Ключевые слова: машинное обучение; программа KNIME; классификация; набор данных; отбор данных; SQL-инъекция; обнаружение угроз в сети; обнаружение подозрительных шаблонов; защита веб-приложений.

Для цитирования: Лапина М.А., Капшук Н.Р., Русанов М.А., Тимофеева Е.Ф. Обнаружение атак с использованием SQL-инъекций по сетевым журналам с использованием методов машинного обучения. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 81–92 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)–6.

1. Introduction

In the modern world, in the age of information technology, databases store a lot of different information: logins and passwords, bank card numbers, home addresses, credit history and much more. Attackers interested in stealing this information are capable of using various types of cyber attacks aimed at hacking servers storing valuable information [1]. One of these methods of cyber attacks is SQL injection. SQL injection is a serious security vulnerability of web applications and systems that have access to databases [2]. The essence of this method is the introduction of malicious SQL code into an Internet resource, which in turn allows attackers to gain access to change data and steal it [3-4]. Usually, this attack is possible when input data is not filtered thoroughly enough when using SQL queries. To successfully neutralize the introduction of malicious code on sites, applications, and ensure the protection of stored information, it is necessary to detect a hacking attempt early and then prevent it. Machine learning in comparison with manual analysis allows you to do this quickly and accurately. This approach not only provides a high level of security, but also provides effective means of detecting, analyzing, and preventing threats [5].

The purpose of the study: creation and identification of the most effective machine learning model for detecting SQL injection into the network in terms of detection accuracy.

To achieve the stated goal of the study, the following tasks were identified: studying the history of the issue, analyzing the parameters of a dataset of SQL attacks, determining input data for machine learning models, creating machine learning models to solve binary classification problems, selecting the most effective models, setting PCA values (principal component analysis), determining the depth

of machine learning, using methods of combating overfitting, building ROC curves and conducting a system analysis of machine learning results.

There are few works with a similar research purpose. One of the differences between this study and others that conduct a comparative analysis of machine learning methods for detecting SQL injections [6-8] is a visual explanation of the implementation of machine learning in the KNIME program and a special methodological toolkit - tree-based machine learning methods, which increases the value of the article.

Among the existing approaches (administrative, legal, and technical), technical ones are used, in particular, using network filters [9].

When solving problems, both general scientific research methods were used: analytical, comparative, and problem-solving methods; and machine learning methods: Decision Tree, Random Forest, Simple Regression Tree, Gradient Boosted Trees, and Tree Ensemble.

Machine learning (ML) is the process of automatically learning and improving the behavior of an artificial intelligence system based on processing an array of training data without explicit programming [10]. In other words, ML is based on finding patterns between incoming features for subsequent prediction.

Analysis of the history of SQL injection attacks showed that they originated in the 1990s, when web applications began to gain popularity. The first such attack was recorded in 1998, highlighting vulnerabilities in database-driven websites that allowed attackers to manipulate SQL queries by injecting malicious code through user input fields [11]. Since then, SQL injection attacks have been used for a quarter of a century. For example, large-scale destructive attacks using SQL injections occurred at Heartland Payment Systems in 2008, when a major payment processor experienced one of the largest data leaks and about 130 million credit and debit card numbers were exposed [12]. In 2011, Sony Pictures was attacked, which compromised Sony's network and digital infrastructure, affecting around 77 million PlayStation Network accounts, costing Sony around $170 million [13].

In 2012, Yahoo Voices suffered a massive data leak, affecting its vast user base and exposing around half a million email addresses and passwords [14].

In 2015, the telecommunications giant TalkTalk suffered a cyberattack, which resulted in almost 157,000 customers losing their personal data [15].

In recent years, the number of cyberattacks has increased many times over [16]. For example, in 2023, in the United States, a group of ransomware attackers successfully injected malicious SQL code into MOVEit Transfer software and a Progress Software product designed to manage file transfers. This attack was only detected a month later [17].

SQL attacks have also been recorded in Russia. For example, in 2024, there was a massive attack on ticket purchasing services. During this period, the total number of simultaneously conducted cyberattacks increased more than 2-fold [18].

Thus, it can be concluded that SQL injections are one of the most common and dangerous methods of attack in the field of cybersecurity, and that in the modern world, network security is of critical importance.

2. Research

During the research, the KNIME platform was used to implement ML and predict potential SQL attacks. KNIME is an environment designed to create algorithms aimed at data analysis and ML, while not requiring full-fledged programming. Unlike other implementation technologies, this program has a user-friendly and flexible interface, and the algorithms are implemented using pre-configured nodes that can be used for built-in data processing pipelines. The advantages of the KNIME program are also open source and its extensibility - the ability to support integration with other platforms such as Python, R, Java and others, and open source code [19].

2.1. Dataset analysis

The dataset chosen for the implementation of the ML was the Web Network dataset [20]. It contains network traffic logs and is designed to classify web requests as "good" or "bad" based on their characteristics. By analyzing patterns in network logs, this dataset helps to identify web requests that can be categorized as "legitimate" or "malicious". It has various HTTP(S) requests, including headers, URLs, and request bodies. Each request is labeled based on its perceived legitimacy. The detection strategy is to identify certain key patterns that indicate an attack [20].

Let us consider the dataset in more detail.

The original dataset contains 522 rows of data and there are 29 columns in total, each of which belongs to a specific data type and contains specific information that can help in training the model. The target column is the "class" column. The ratio of safe requests to suspicious ones in this column is 321 to 201. In the course of the study, a comparative analysis of the web network dataset was carried out (Table 1).

The dataset [20] contains 12 duplicate columns with similar data. Repeated recordings can lead to incorrect operation of the models. To solve this problem, duplicate columns were not used as input data. There are also various concepts such as "method", "path", "features" and "prediction". In the "method" column only 22% of the cells have the values "GET" or "POST", and the rest contain zeros, which indicates that this column has incorrect values, so it should be excluded. The "path" column, contains various query paths, but it does not have information about the contents of the queries, since information about the number of special characters has been placed in separate columns. The "features" column contains arrays with the same data that were presented in the previous columns. And "prediction" is the results of a study conducted by the author of the dataset. Thus, we can conclude that for the most efficient ML, the following columns will be used as input data for the models: "single_q_1", "double_q_2", "dashes_3", "braces_4", "spaces_5", "percentages_6", "semicolons_7", "angle_brackets_8", "special_chars_9", "path_length_10", "body_length_11", "badwords_count_12" and "class". It is important to keep in mind that the "class" column is the target column. No transformations will occur with it, and it will be used to compare the values contained in it with the values that will be output by ML models in order to obtain a percentage of prediction accuracy.

2.2. Modeling

After selecting the input data, the ML models were constructed (Fig. 1).

The justification for the order of connecting the nodes of the generalizing model is determined by the analysis (Table 2).

A total of 5 models were created, each using its own training method. Among them: Decision Tree, Random Forest, Simple Regression Tree, Gradient Boosted Trees, Tree Ensemble. The selected machine learning methods related to trees are characterized by high accuracy, interpretability, and they can also be used to work with missing values.

It should be noted that with ML for the same model and with the same settings, different accuracy results could be obtained, and the error was about 1-2%. To get more objective results, training was performed 10 times for each ML model setting option, after which the average value between the results was returned.

With the initial (default) settings, each model showed the following accuracy results (Table 3).

In order to improve the accuracy of the models, it is necessary to perform additional adjustments. One of these adjustments is the PCA adjustment. PCA (principal component analysis) is a statistical method that allows reducing the dimensionality of data (Table 4), while preserving the greatest amount of information [21]. The values were adjusted in the PCA block.

The accuracy of the models from the PCA value is presented more clearly below (Fig. 2).

Table 1. Analysis of data from the Web Network dataset.

Name of the columns	Data type	Description
method	Categorical	Indicates the type of operation the user wants to perform (GET or POST).
path	Text	Request path
single_q	Quantitative	Number of single quotes in the query (‘)
double_q	Quantitative	Number of double quotes in the query (“)
dashes	Quantitative	Number of dashes in the query (-)
braces	Quantitative	Number of curly braces in the query ({})
spaces	Quantitative	Number of spaces in the query
percentages	Quantitative	Number of percent characters in the query (%)
semicolons	Quantitative	Number of semicolons in the query (;)
angle_brackets	Quantitative	Number of angle brackets (< >)
special_chars	Quantitative	Number of special characters in the query
path_length	Quantitative	Request path length
body_length	Quantitative	Length of the request body. Only available when using the POST method in the "method" column.
badwords_count	Quantitative	The number of suspicious words that may be frequently used in SQL injections.
single_q_1	Quantitative	Similar to "single_q"
double_q_2	Quantitative	Similar to "double_q"
dashes_3	Quantitative	Similar to "dashes"
braces_4	Quantitative	Similar to "braces"
spaces_5	Quantitative	Similar to "spaces"
percentages_6	Quantitative	Similar to "percentages"
semicolons_7	Quantitative	Similar to "semicolons"
angle_brackets_8	Quantitative	Similar to "angle_brackets"
special_chars_9	Quantitative	Similar to "special_chars"
path_length_10	Quantitative	Similar to "path_length"
body_length_11	Quantitative	Similar to "body_length"
badwords_count_12	Quantitative	Similar to "badwords_count"
class	Binary	Mark what the request is (0 – safe, 1 – suspicious). This is also the target column, since it will be compared with the predicted data.
features	Text	The previously specified cell values combined into a list.
prediction	Binary	Prediction of the previously used learning model.

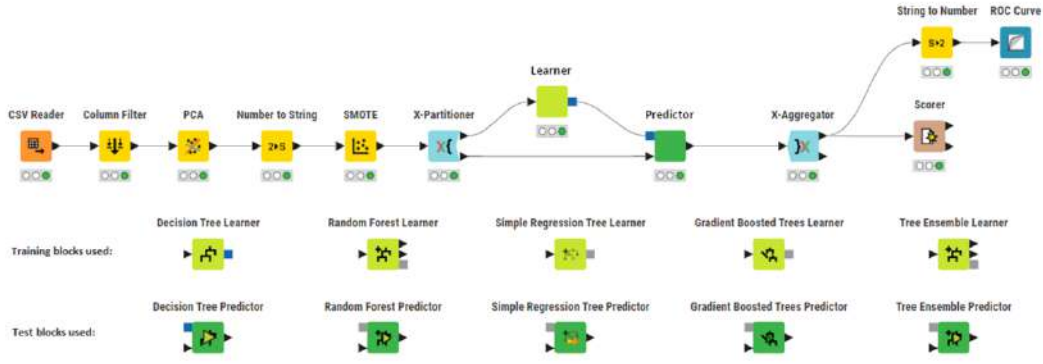


Fig 1. Generalized Machine Learning Model and Machine Learning Blocks in KNIME.

Table 2. Purpose of nodes in each model.

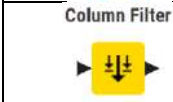
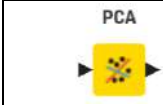
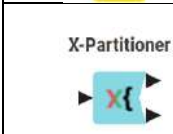
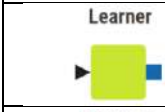
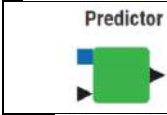
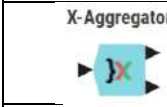
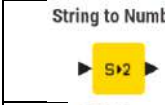
Node	Node name	Purpose of the node
	CSV Reader	Uploads a CSV file.
	Column Filter	Discards the specified columns.
	PCA	Reduces the dimensionality of data.
	Number to String	Converts numeric data to strings.
	SMOTE	Makes the dataset balanced.
	X-Partitioner	Splits the data into a set number of parts and performs cross-validation. During the research, the data is divided into five parts, after which, during each iteration, four parts are sent for training and one for testing.
	Learner	Learning node.
	Predictor	Testing node.
	X-Aggregator	Returns the average accuracy value of the results at each iteration after cross-validation.
	Scorer	Outputs the accuracy of the model.
	String to Number	Converts string data to numbers.
	ROC Curve	Builds a ROC curve.

Table 3. Accuracy of machine learning models on initial settings.

Machine learning method	Accuracy (in %)
Decision Tree	93,730
Random Forest	93,502
Simple Regression Tree	94,199
Gradient Boosted Trees	93,729
Tree Ensemble	94,008

Table 4. Dependence of model accuracy on the PCA block value.

PCA Block Value	Model accuracy (in %)				
	Decision Tree	Random Forest	Simple Regression Tree	Gradient Boosted Trees	Tree Ensemble
1	93,730	93,502	94,199	93,729	94,008
2	94,131	94,617	93,868	93,903	94,617
3	94,814	94,706	94,581	94,461	94,531
4	95,053	95,054	95,229	95,611	95,211
5	96,025	96,342	96,080	96,064	96,569
6	96,271	97,039	**97,038**	96,950	97,073
7	96,236	97,126	96,848	96,656	97,144
8	96,273	97,231	96,395	96,672	97,214
9	96,169	**97,598**	96,480	96,811	**97,562**
10	**96,533**	97,195	96,672	96,882	97,231
11	96,427	97,160	96,550	**97,125**	97,336
12	96,341	97,108	96,741	97,089	97,370

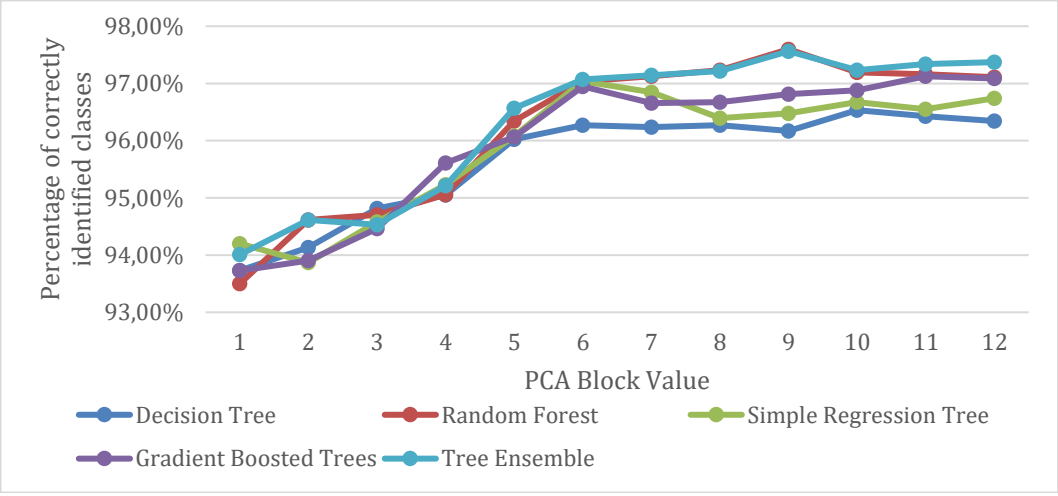


Fig. 2. Dependence of the accuracy of the models on the value of the PCA block.

Table 5. Dependence of model accuracy on learning depth.

Learning depth	Model accuracy (in %)				
	Decision Tree	Random Forest	Simple Regression Tree	Gradient Boosted Trees	Tree Ensemble
1	96,307	91,516	92,754	96,707	90,485
2	96,290	96,467	95,958	96,828	95,785
3	96,707	97,073	96,482	96,879	96,690
4	**96,898**	97,143	96,342	96,864	96,795
5	96,464	97,546	96,550	**97,003**	97,038
6	96,237	**97,580**	96,255	96,483	97,283
7	95,924	**97,580**	**96,618**	96,306	97,213
8	96,133	97,545	96,534	96,725	97,387
9	95,890	97,300	96,498	96,637	**97,473**
10	96,081	97,421	96,603	96,830	97,457

From these results, we can conclude that for a model with the "Decision Tree" learning method, the most optimal reduction is to 10 columns, for the "Random Forest" and "Tree Ensemble" methods - to 9 columns, for the "Simple Regression Tree" – to 6 columns, and for the "Gradient Booste Trees - to 11 columns (Fig. 2). For each model, the learning depth – number of hidden learning layers [22] – was adjusted with the numbers of columns that were obtained before (Table 4). The depth of learning was adjusted in the Learner block.

The results showed that the Decision Tree model achieves the best result with 4 training layers, the Random Forest model with 6-7 training layers, the Simple Regression Tree with 7 training layers, the Gradient Boosted Trees with 5 training layers, and the Tree Ensemble with 9 training layers (Fig. 3).

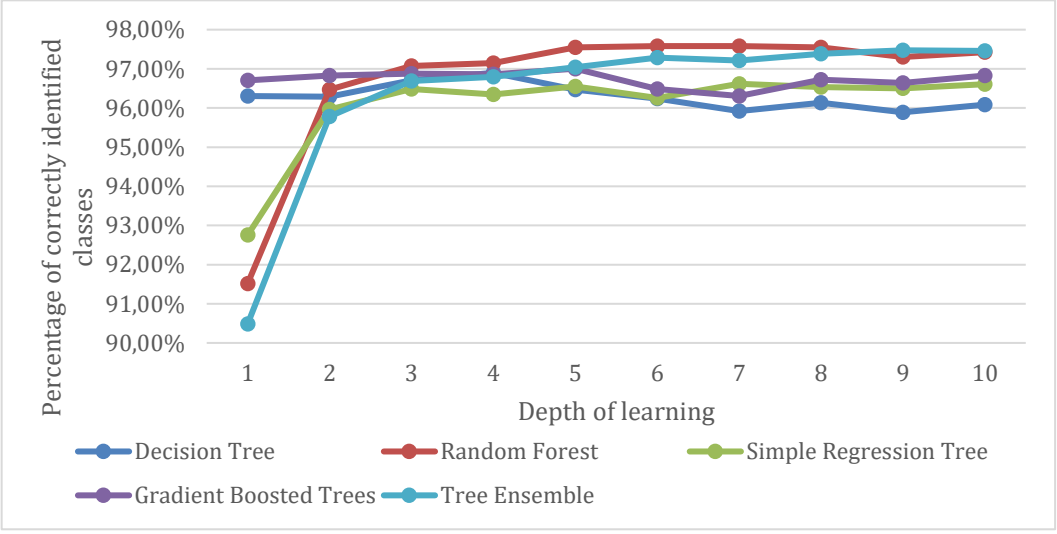


Fig. 3. Dependence of model accuracy on learning depth.

2.3. Combating overfitting

It is important to consider that during the training of neural networks, so-called overfitting may occur, which can lead to worse forecasting results. This phenomenon occurs when the model adjusts too much to the training data, which is why it begins to work poorly with new data. Combating overfitting is an integral task in the field of ML. One of the methods for solving this problem is cross-validation - a method for assessing the quality of a model by dividing the data into several parts, after which the model is trained and predicted on different subsets of data. In this case, the dataset [20] is divided into five equal parts using the "X-partitioner" block. The model is then trained through five iterations, where one part of the data is used in testing and the rest in training. After that, the "X-aggregator" block returns the average accuracy value. To clearly see the forecasting quality of each model, it is necessary to use ROC curves (receiver operating characteristic) - graphs that are used to assess the quality of binary classifiers [23]. An important part of the ROC curve is the area under it, where a value of "1" indicates an ideal classifier, and a value of "0.5" indicates a large amount of randomness during forecasting. Based on the ROC curve graphs (Fig. 4), it can be concluded that the resulting ML models were quite effective in predicting the potential introduction of malicious SQL code into the network. The resulting models showed high efficiency in detecting patterns between input features and subsequent prediction, as evidenced by the results.

Based on all the obtained results, a comparative table of machine learning methods was compiled (Table 6).

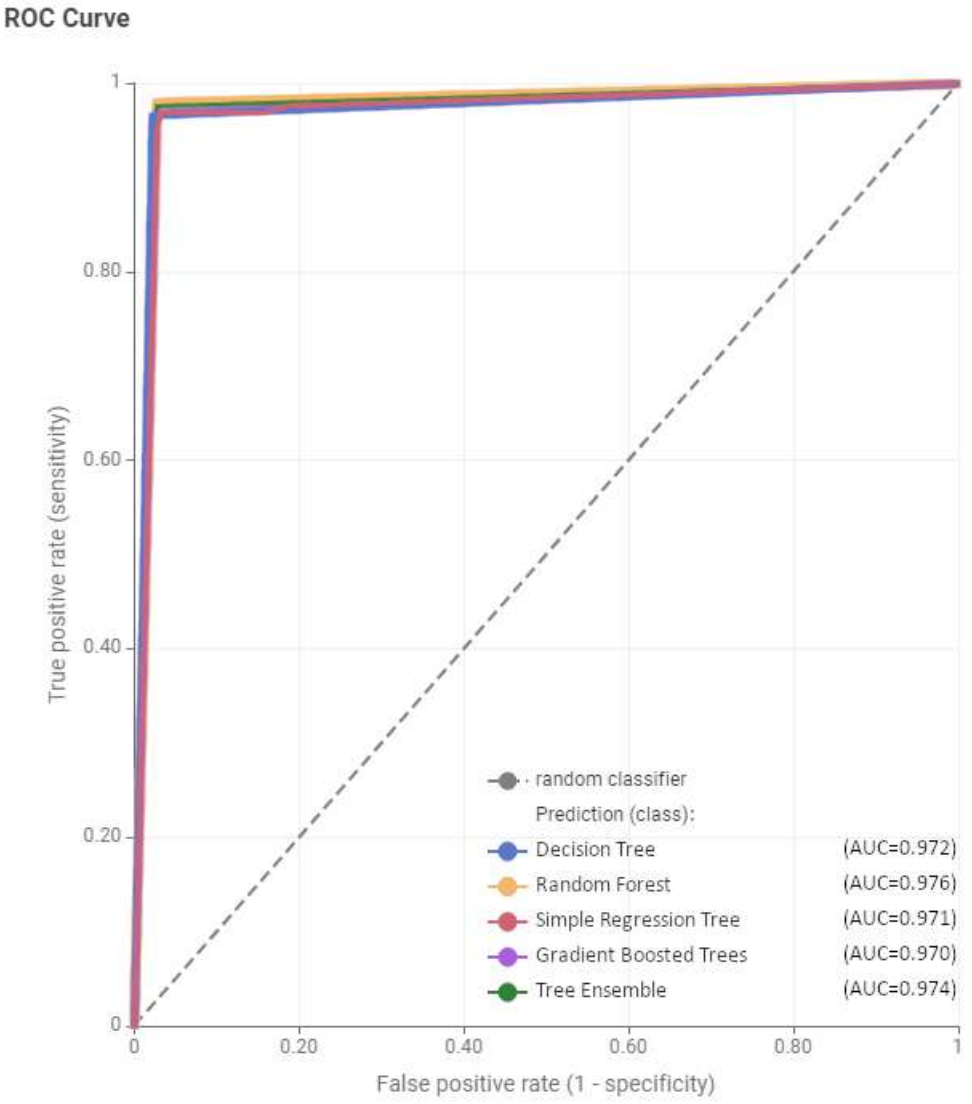


Fig. 4. ROC curve graphs.

Table 6. Best results of the models.

Machine learning method	PCA Block Value	Learning depth (number of learning layers)	Highest accuracy (in %)	AUC (Area Under the Curve)
Decision Tree	10	4	96,898	0,972
Random Forest	**9**	**6-7**	**97,580**	**0,976**
Simple Regression Tree	6	7	96,618	0,971
Gradient Boosted Trees	11	5	97,003	0,970
Tree Ensemble	9	9	97,473	0,974

2.4. Analysis of the results

The general characteristics of the models as a whole indicate the high efficiency of each of the ML methods used. The analysis of accuracy and ROC curves shows that a model using a Random Forest as a machine learning method provides the highest quality of binary classification. Optimization of the model parameters to achieve maximum classification accuracy was achieved by adjusting the values of the PCA block and the depth (number of layers) of the ML.

3. Conclusion

The conducted research on the creation of machine learning models to detect the introduction of malicious SQL code into the network differed from similar studies of this problem by implementing models in the KNIME program. An analysis of other work related to the detection of potential SQL injections using ML was carried out. The data set was analyzed, and ML tree-based models were compiled. Each ML method used was configured in such a way as to increase the percentage of correctly identified classes, and a comparative analysis was performed. The "Random Forest" model showed the best result with the highest accuracy of 97.58%, and the area under the ROC curve graph compiled to assess the quality of this model is 0.976. Thus, the ML in the KNIME program allows you to create effective models for detecting the potential introduction of malicious SQL code into the network.

References

[1]. "Stab me if you can" – how websites and SQL databases are attacked with injections – Dmitry Ushakov on TenChat.ru. URL: https://tenchat.ru/media/2607916-protkni-menya-yesli-smozhesh--kak-atakuyut-vebsayty-i-bazy-dannykh-sql-inyektsiyami (date of access: 17.04.2025).

[2]. Khomyarchuk M. V. Modern trends and innovations in web security: challenges, solutions and prospects //Science and modern education: current issues. – 2023. – p. 28.

[3]. Oglov V. A. Investigation of sql injection attacks and analysis of web site security //Bulletin of the Magistracy. - 2024. – p. 15.

[4]. Manukyan A. R. Problems of ensuring cybersecurity at the present stage //Law and management. – 2024. – No. 10. – pp. 313-316.

[5]. Peev D. D., Pankov K. N. The use of computer vision and machine learning technologies in the field of secure information systems //Signal synchronization, generation and processing systems. – p. 28.

[6]. Yudova E. A., Laponina O. R. Comparative analysis of approaches to detecting SQL injections using machine learning methods //International Journal of Open Information Technologies. - 2023. - Vol. 11. - No. 6. - pp. 175-181.

[7]. Kasim Ö. An ensemble classification-based approach to detect attack level of SQL injections //Journal of Information Security and Applications. – 2021. – T. 59. – C. 102852.

[8]. Erdődi L., Sommervoll Å. Å., Zennaro F. M. Simulating SQL injection vulnerability exploitation using Q-learning reinforcement learning agents //Journal of Information Security and Applications. – 2021. – T. 61. – C. 102903.

[9]. Zaozersky A. A. Technical approaches to information protection //BBK 1 N 34. – P. 6505.

[10]. Chesalov A. Y. Glossary on artificial intelligence: 2500 terms/ A. Y. Chesalov - "Publishing solutions", 2022. - 670 p.

[11]. SQL attack. URL: https://ru.easiio.com/sql-attack/ (date of access: 03.04.2025).

[12]. The Hearland Breach | A cautionary Tale foe E-Commerce. URL: https://blog.comodo.com/e-commerce/the-heartland-breach-a-cautionary-tale-for-e-commerce/ (date of access: 03.04.2025).

[13]. Indonesian Journal of Electrical Engineering and Computer Science Vol. 21, No. 2, February 2021, pp. 1121-1131.

[14]. Yahoo Hack Leaks 453,000 Voices Passwords. URL: https://www.darkreading.com/cyberattacks-data-breaches/yahoo-hack-leaks-453-000-voice-passwords (date of access: 03.04.2025).

[15]. Unknown persons hacked the British TalkTalk provider – Xakep. URL: https://xakep.ru/2015/10/27/talktalk-hacked/ (date of access: 03.04.2025).

[16]. Nathan C., Steven F., Human Aspects of Information Security and Assurance, p.329, New York: Springer International Publishing (2022).

[17]. Current threats: The second quarter of 2023. URL: https://www.ptsecurity.com/ru-ru/research/analytics/cybersecurity-threatscape-2023-q2/ (date of access: 03.04.2025).
[18]. Major cyber attacks and leaks in Russia in 2024. URL: https://blog.cortel.cloud/2024/05/23/krupnye-kiberataki-i-utechki-pervoj-poloviny-2024-goda-v-rossii/?ysclid=m929qx878m857705097 (date of access: 03.04.2025).
[19]. KNIME Analytics Platform | KNIME. URL: https://www.knime.com/knime-analytics-platform (date of access: 15.05.2025).
[20]. Web Network. URL: https://www.kaggle.com/datasets/willianoliveiragibin/web-network (date of access: 21.03.2025).
[21]. How to use the PCA method to reduce the dimension of data / Habr. URL: https://habr.com/ru/companies/otus/articles/769274 / (date of access: 04.03.2025).
[22]. Machine Learning Glossary | Google for Developers. URL: https://developers.google.com/machine-learning/glossary#d (date of access: 15.05.2025).
[23]. Kostromitin M. A. The fight against retraining of neural networks: causes, effects and methods of prevention //BBK 1 N 34. - p. 2809.

Information about authors

Мария Анатольевна ЛАПИНА – кандидат физико-математических наук, доцент кафедры вычислительно математики и кибернетики факультета математики и компьютерных наук имени профессора Н.И. Червякова Северо-Кавказского федерального университета. Сфера научных интересов: цифровые технологии, анализ данных, искусственный интеллект, кибербезопасность, управление информационной безопасностью, криптография.

Maria Anatolyevna LAPINA – Cand. Sci. (Phys.-Math.), Associate Professor at the Department of Computational Mathematics and Cybernetics, Faculty of Mathematics and Computer Science named after Professor N.I. Chervyakov, North Caucasus Federal University. Research interests: digital technologies, data analysis, artificial intelligence, cybersecurity, information security management and cryptography.

Николай Романович КАПШУК – студент кафедры вычислительно математики и кибернетики факультета математики и компьютерных наук имени профессора Н.И. Червякова Северо-Кавказского федерального университета. Сфера научных интересов: информационная безопасность, технологии сетевой безопасности, машинное обучение, нейронные сети, цифровые технологии.

Nikolay Romanovich KAPSHUK – student at the Department of Computational Mathematics and Cybernetics, Faculty of Mathematics and Computer Science named after Professor N.I. Chervyakov, North Caucasus Federal University. Research interests: information security, network security technologies, machine learning, neural networks, digital technologies.

Михаил Андреевич РУСАНОВ – аспирант Института информационных технологий, Московский финансово-юридический университет. Сфера научных интересов: информационная безопасность, управление информационной безопасностью, машинное обучение, нейронные сети, обнаружение аномалий.

Mikhail Andreevich RUSANOV – postgraduate student at the Institute of Information Technology, Moscow University of Finance and Law. Research interests: information security, information security management, machine learning, neural networks, anomaly detection.

Елена Федоровна ТИМОФЕЕВА – доцент кафедры математического анализа алгебры и геометрии факультета математики и компьютерных наук имени профессора Н.И. Червякова Северо-Кавказского федерального университета. Сфера научных интересов: математическое моделирование, численные методы, задачи гидродинамики.

Elena Fedorovna TIMOFEEVA – Associate Professor of the Department of Mathematical Analysis of Algebra and, Faculty of Mathematics and Computer Science named after Professor N.I. Chervyakov, North Caucasus Federal University. Research interests: mathematical modeling, numerical methods, problems of hydrodynamics.

DOI: 10.15514/ISPRAS-2025-37(5)-7

The Dynamic Adaptive Packet Buffering (DAPB) Algorithm for Service Mesh Performance Enhancement Based on eBPF

H-D. Djambong Tenkeu, ORCID: 0009-0002-4689-1665 <Dzhambong.T.K@hse.ru>
D.V. Alexandrov, ORCID: 0000-0002-9759-8787 <dvalexandrov@hse.ru>
National Research University "Higher School of Economics"
11, Pokrovsky blvd, Moscow, 109028, Russia.

Abstract. This paper introduces the Dynamic Adaptive Packet Buffering (DAPB) algorithm. Designed to enhance data transfer efficiency in modern networking environments, it is built on the principles of Nagle's algorithm. DAPB addresses the limitations of existing buffering techniques by dynamically adjusting its behavior based on real-time network conditions, application requirements, and latency sensitivity. The algorithm incorporates context-sensitive buffering, adaptive timeout mechanisms, and machine learning-driven predictions to achieve a balance between efficiency, latency, and energy consumption. DAPB's context-aware buffering tailors its strategy to the specific needs of the application, minimizing buffering for latency-sensitive applications like VoIP and online gaming, while maximizing buffering for throughput-sensitive applications like file transfers. The adaptive timeout mechanism dynamically adjusts the waiting timeout based on network conditions such as round-trip time, packet loss, and jitter, ensuring optimal performance under varying workloads. Machine learning models are used to predict optimal buffer sizes and timeout values, leveraging historical data and real-time metrics to improve decision-making. The algorithm also features selective aggregation, intelligently deciding which packets to aggregate and which to send immediately. This ensures that urgent packets are transmitted without delay, while nonurgent packets are aggregated to reduce overhead. Additionally, DAPB prioritizes energy efficiency by optimizing buffer sizes and timeout values, making it suitable for energy-constrained environments like edge computing and IoT devices. The DAPB algorithm is expected to improve the data transfer performance in various scenarios. Compared to the standard Nagle algorithm, the DAPB algorithm is expected to reduce latency, improve throughput, and enhance energy efficiency. This paper is the result of a research project implemented as part of the Basic Research Program at the National Research University Higher School of Economics (HSE University).

Keywords: dynamic adaptive packet buffering (DAPB); extended Berkeley packet filter (eBPF); kernel-level packet processing; service mesh

For citation: Djambong Tenkeu H-D., Alexandrov D.V. The Dynamic Adaptive Packet Buffering (DAPB) Algorithm for Service Mesh Performance Enhancement Based on eBPF. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 93-110. DOI: 10.15514/ISPRAS-2025-37(5)-7.

Алгоритм динамической адаптивной буферизации пакетов (DAPB) для повышения производительности Service Mesh на основе eBPF

Х-Д. Джамбонг Тенке, ORCID: 0009-0002-4689-1665 <Dzhambong.T.K@hse.ru>
Д.В. Александров, ORCID: 0000-0002-9759-8787 <dvalexandrov@hse.ru>

Национальный исследовательский университет «Высшая школа экономики», Россия, 109028, г. Москва, ул. Покровский бульвар, д. 11.

Аннотация. Данная статья представляет алгоритм динамической адаптивной буферизации пакетов (Dynamic Adaptive Packet Buffering, DAPB). Разработанный для повышения эффективности передачи данных в современных сетевых средах, алгоритм основан на принципах алгоритма Нейгла. DAPB преодолевает ограничения существующих методов буферизации за счет динамической адаптации поведения на основе текущих сетевых условий, требований приложений и чувствительности к задержкам. Алгоритм сочетает контекстно-зависимую буферизацию, адаптивные механизмы таймаутов и прогнозирование на основе машинного обучения для оптимального баланса между эффективностью, задержкой и энергопотреблением. Контекстно-ориентированная буферизация адаптирует стратегию под конкретные приложения: минимизирует буферизацию для чувствительных к задержкам сервисов (VoIP, онлайн-игры) и максимизирует для throughput-ориентированных задач (передача файлов). Адаптивный механизм таймаутов динамически регулирует период ожидания с учетом времени кругового обхода (RTT), потерь пакетов и джиттера, обеспечивая оптимальную производительность при изменяющейся нагрузке. Модели машинного обучения предсказывают оптимальные размеры буфера и значения таймаутов, используя исторические данные и метрики реального времени. Алгоритм реализует селективную агрегацию пакетов, интеллектуально определяя какие пакеты следует агрегировать, а какие передавать немедленно. DAPB уделяет особое внимание энергоэффективности за счет оптимизации параметров буферизации, что делает его применимым в энергоограниченных средах (edge computing, IoT устройства). По сравнению со стандартным алгоритмом Нейгла, DAPB демонстрирует снижение задержек, увеличение пропускной способности и улучшение энергоэффективности. Исследование выполнено в рамках Программы фундаментальных исследований Национального исследовательского университета "Высшая школа экономики" (НИУ ВШЭ).

Ключевые слова: динамическая адаптивная буферизация пакетов (DAPB); расширенный фильтр пакетов Беркли (eBPF); обработка пакетов в ядре; service mesh

Для цитирования: Джамбонг Тенке Х-Д., Александров Д.В.. Алгоритм динамической адаптивной буферизации пакетов (DAPB) для повышения производительности Service Mesh на основе eBPF. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 93–110 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)-7.

1. Introduction

The proliferation of micro-services as the de-facto standard for building scalable and resilient applications has necessitated the evolution of underlying infrastructures that can adeptly manage the complexities of distributed systems. Service meshes have emerged as a critical component in the cloud-native ecosystem, offering a dedicated infrastructure layer that simplifies inter-service communication, enforces security policies, and provides observability across microservices. Istio, a leading service mesh implementation, exemplifies this by deploying a sidecar proxy alongside each microservice, thus abstracting the intricacies of network management from the application logic.

Despite the advantages conferred by service meshes, they are not without their challenges. The introduction of an intermediary proxy layer, while beneficial for manageability and control, inadvertently introduces additional overheads (like higher latency) in the communication path. These overheads are particularly pronounced within the Linux kernel network stack, where packet transmission is subject to several context switching and kernel-space to user-space communication.

As microservices continue to scale and the demand for low-latency, high-throughput systems grow, the need to address these overheads becomes increasingly critical.

One of the main approaches to solving these issues consists of performing traffic buffering. It allows optimizing data transmission and managing network congestion. One of the most popular buffering algorithms is Nagle's algorithm, introduced by John Nagle in 1984. Improves TCP communication by reducing the transmission of small packets over networks [1]. Designed to address inefficiencies caused by applications sending frequent, tiny data bursts, it mitigates network congestion by temporarily buffering small writes until either an acknowledgment (ACK) is received for previous data or enough data accumulates to form a full TCP segment. The algorithm ensures that only one small packet remains unacknowledged at a time, preventing the network from being flooded with tiny packets.

Although effective for bulk data transfers, Nagle's algorithm can introduce latency in interactive applications like gaming or SSH due to its interaction with TCP's delayed ACK mechanism, which waits up to 200 milliseconds to combine ACKs with outgoing data. This trade-off led to criticism [2]. The main point is that the strength of this algorithm (reducing small packets) is also its weakness. Modern systems often disable it for latency-sensitive applications (e.g., VoIP) using the TCP_NODELAY socket option, but it remains valuable for optimizing high-throughput workloads like file transfers.

This paper introduces a novel algorithm to improve data transfer efficiency by dynamically adapting to real-time network conditions and application needs. It does so through context-sensitive buffering, adaptive timeouts, and machine learning. Such an algorithm shall help improving performance, namely reducing latency, and improve energy efficiency in modern networking environments.

The remainder of the paper is organized as follows. Section 2 discusses the related works, while Section 3 presents the background and motivation of the new algorithm. Section 4 describes the new algorithm. Section 5 defines the performance metrics that can be used to assess the performance of the new algorithm. Section 6 contains the risks and limitations of the DAPB algorithm. Section 7 specifies the next steps of this research.

2. Related works

eBPF (extended Berkeley Packet Filter) enables the execution of user-defined programs within the Linux kernel. Its lineage begins with the Berkeley Packet Filter (BPF), introduced in 1993 by McCanne and Jacobson as a mechanism to efficiently capture network packets in user space [3]. Classic BPF (cBPF) employed a simple register-based virtual machine to execute filter programs in the kernel, reducing unnecessary data copying between the kernel and the user space.

The transition to eBPF began in 2014 with its integration into Linux kernel 3.18. This overhaul, led by Alexei Starovoitov, reimagined BPF as a general-purpose execution environment [4]. Key enhancements included a 64-bit register model, a Just-In-Time (JIT) compiler, and a richer instruction set, enabling eBPF programs to interact safely with kernel data structures. The most transformative applications of eBPF have emerged in networking. The 2018 introduction of XDP (eXpress Data Path) [5] marked a paradigm shift by enabling packet processing at the driver layer, bypassing the kernel network stack entirely.

Network congestion is a common issue in computer network engineering. To solve it, several buffering algorithms and techniques were created. The Sliding Window Protocol, as described in [6], is fundamental to TCP's flow control, allowing multiple packets to remain in transit before requiring acknowledgments. This approach maximizes throughput while preventing receiver overload by dynamically adjusting the window size based on network conditions.

Modern congestion control algorithms such as TCP BBR ([7] & [8]) represent another category, using bandwidth and latency measurements to dynamically optimize transmission rates. Meanwhile, at the hardware level, Direct Memory Access (DMA) [9] and Zero-Copy Buffering [10] minimize

CPU involvement by enabling direct data transfers between devices and memory, significantly reducing latency in high-speed networks.

Nagle's algorithm reduces TCP overhead by buffering small writes until either: (1) enough data accumulate to fill a packet or (2) all sent data are acknowledged. Although it minimizes "tinygrams" that waste bandwidth, it can increase latency [6], prompting many real-time systems to disable it via 'TCP_NODELAY'. The algorithm remains fundamental in throughput-latency tradeoff studies.

Compared to existing network (Table 1) enhancement algorithms, the DAPB algorithm introduces several novel improvements. Context-sensitive adaptability and machine learning-driven optimization as key novelties. Unlike traditional Nagle's algorithm, which uses fixed rules, DAPB dynamically adjusts buffer sizes and timeout mechanisms based on real-time network conditions. These conditions include (but are not limited to) packet round-trip time (RTT), the variability in packet arrival times (known as jitter), and the packet loss. The DAPB algorithm also considers application requirements (e.g. latency sensitivity). Using machine learning, it can predict optimal configurations [11], ensuring better performance in diverse scenarios. Additionally, DAPB incorporates selective buffering to prioritize urgent packets, reducing latency for real-time applications, while optimizing energy efficiency for resource-constrained environments like IoT. This makes DAPB more versatile, efficient, and adaptive than static or rule-based algorithms.

Table 1: Comparative analysis of buffering algorithms.

Characteristic	Nagle's	BBR	Circular	QUIC	DAPB
Adaptivity	Fixed	Congestion	Fixed	Per-conn	ML-driven
Latency	Poor	Moderate	Low	Excellent	Context
Throughput	Moderate	High	High	High	Adaptive
Energy	None	Partial	None	None	Optimized
Protocol	TCP	Transport	Generic	QUIC	Multi-protocol
Layer	Kernel	CC	User	User	Kernel + eBPF
ML	No	No	No	No	Yes
Priority	None	None	None	Stream	Urgency
Dynamic	No	RTT	No	Per-flow	RTT + Jitter + Packet loss
HW Accel	No	No	No	No	Partial

- Comparison includes classic (Nagle's, Circular) and modern (BBR, QUIC) approaches
- DAPB introduces ML-driven adaptivity and multi-protocol support
- Evaluated for cloud-native service mesh requirements

3. Background and motivation

3.1 Flow of traffic within the service mesh and Linux operating system

In a service mesh architecture, communication between application components occurs through a dedicated infrastructure layer composed of programmable proxies (Fig. **1**). These proxies, deployed as sidecars (e.g., Envoy, Linkerd-proxy), run alongside application containers in user space. Instead of applications directly managing network logic, they delegate tasks like service discovery, retries, or mutual Transport Layer Security (mTLS) to their sidecars via local inter-process communication (IPC) mechanisms such as Unix domain sockets.

The service mesh divides responsibilities between a control plane (e.g., Istio Pilot [12], Linkerd's control plane) and a data plane (sidecar proxies). The control plane acts as a centralized orchestrator, distributing policies, routing rules, and security configurations (e.g., certificates for mTLS) to data plane proxies. These proxies enforce rules at the application layer (Layer 7), enabling features like HTTP/2-based load balancing, circuit breaking, and header-based routing. Unlike the Linux kernel's IP/TCP-centric approach, service meshes prioritize protocols like HTTP, gRPC, or service-specific APIs.

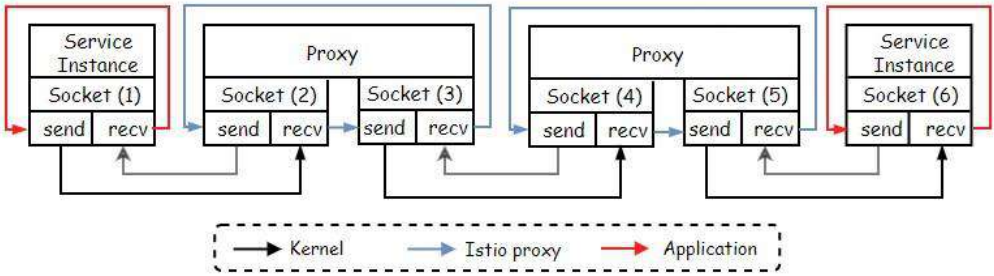


Fig. 1. Traffic flow within Istio service mesh.

Proxies intercept traffic using mechanisms like iptables rules or eBPF programs to redirect packets to the sidecar before reaching the application. For example, in Kubernetes, an init container may configure networking rules to ensure that all ingress/egress traffic flows through the proxy.

Within the Linux operating system (Fig. 2), data travels across multiple layers with distinct responsibilities. Applications in user space initiate communication using programming interfaces like sockets and system calls. For example, a web browser might use TCP socket functions from the standard C library to send HTTP requests. These requests get passed to the kernel via syscalls like sendto() or write(), which transition the execution from user mode to kernel mode.

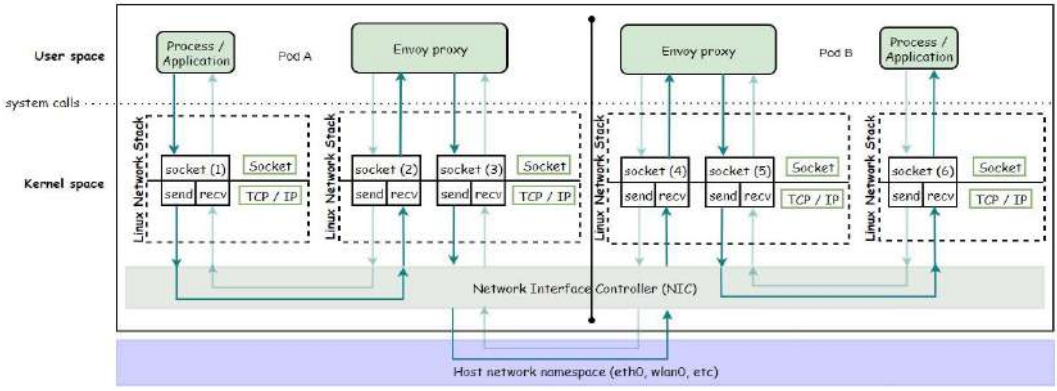


Fig. 2. Traffic flow in Linux OS running within a service mesh.

3.2 Nagle's algorithm

Nagle's algorithm is used to optimize TCP communication by decreasing the number of small packets transmitted over the network [12]. Introduced by John Nagle in 1984, it is particularly effective in situations where applications frequently send small amounts of data. Its main objective is to reduce the overhead that comes with sending numerous small packets, which can contribute to network congestion and inefficient bandwidth usage. Here is how it works:

- Buffering Small Packets: When an application transmits a small amount of data (less than the Maximum Segment Size (MSS), Nagle's algorithm temporarily stores those data in a buffer instead of sending them immediately as a separate packet. This design reduces overhead by minimizing the number of small packets, a trade-off between latency and efficiency noted in [14] and [15].
- Combining Packets: The algorithm waits for one of the following conditions to occur:
  - ✓ An acknowledgment (ACK) from the receiver for data that has already been sent, or
  - ✓ Additional data from the application that can be combined with the buffered data.

This approach, while effective for bulk data transfers, can introduce undesirable latency for interactive applications, as observed in [16] and [17].

- Sending Large Packets: Once one of these conditions is satisfied, the algorithm transmits the buffered data along with any new data as a single larger packet. This optimization leverages network efficiency by amortizing per-packet overhead, a principle further analyzed in [18] and [19].

3.3 The eBPF Technology

eBPF (extended Berkeley Packet Filter) is a Linux kernel innovation that enables developers to run custom, event-driven programs securely within the kernel space without modifying kernel source code or rebooting the system. Originally designed for network packet filtering, eBPF has evolved into a versatile framework to improve performance, observability, and security in modern computing environments [20].

eBPF programs are executed in a sandbox environment, ensuring safety by verifying the code before execution to prevent crashes or resource leaks. These programs attach to predefined hooks in the kernel, such as network events, system calls, or function entries/exits, allowing real-time data processing. For example, eBPF can intercept network packets to optimize routing, monitor application behavior for debugging, or enforce security policies by auditing system activity. It offers the following key advantages:

- Performance: by operating in-kernel, eBPF minimizes context switches and data copying, reducing overhead for tasks like packet processing or monitoring.
- Flexibility: developers can dynamically load programs to adapt to changing needs, such as scaling service mesh traffic or troubleshooting latency.
- Safety: a built-in verifier ensures that programs run without destabilizing the kernel, enforcing strict rules on memory access and loop structures.

To understand eBPF, it is essential to explore its core concepts, including program types, maps, and specialized frameworks such as XDP.

3.3.1 eBPF program types

eBPF supports a variety of program types, each designed for specific use cases. These program types determine where and how eBPF programs can be attached within the kernel. Some common eBPF program types include the following:

- Socket Filtering: Used for filtering and processing network packets at the socket level.
- Kprobes and Uprobes: allow tracing of kernel and user-space functions, respectively, for debugging and observability.
- Tracepoints: Attach to predefined kernel tracepoints to monitor system events.
- XDP (eXpress Data Path): a high-performance program type for processing network packets at the earliest possible point in the kernel's networking stack.
- TC (Traffic Control): used for advanced packet processing and traffic shaping in the kernel's networking subsystem.
- Perf Events: Enable monitoring of hardware and software performance events.

3.3.2 eBPF Maps

eBPF maps are key-value data structures that allow eBPF programs to store and share data between user space and kernel space, or between multiple eBPF programs. They are a fundamental building

block for creating complex and stateful eBPF applications. Common types of eBPF maps include the following:

- Hash Maps: store key-value pairs in a hash table for efficient lookups.
- Array Maps: use integer keys to store fixed-size values, providing fast access.
- Per-CPU Maps: maintain separate data for each CPU core, great for high-performance use cases.
- Ring buffer: a high-throughput data structure for passing data between eBPF programs and the user space.
- LRU (Least Recently Used) Maps: automatically evict least recently used entries to manage memory efficiently.

Maps enable eBPF programs to maintain state, aggregate data, and communicate with user space applications, making them indispensable for advanced use cases like network monitoring and security enforcement.

3.3.3 eXpress Data Path (XDP)

XDP is a high-performance eBPF program type designed to process network packets at the earliest possible point in the kernel's networking stack, often before they reach the kernel's network layer. This makes XDP ideal for use cases requiring low-latency packet processing, such as:

- DDoS mitigation: dropping malicious packets before they consume system resources.
- Load balancing: distributing network traffic across multiple servers with minimal overhead.
- Packet filtering: implementing custom filtering logic at line rate.
- Protocol parsing: extracting and processing custom protocol headers efficiently.

XDP programs are typically attached to network interfaces and operate in one of three modes:

- Native Mode: runs the XDP program directly on the network interface card (NIC) driver.
- Off-loaded mode: offloads the XDP program to the NIC hardware for maximum performance.
- Generic Mode: runs the XDP program in the kernel as a fallback when hardware offloading is not available.

3.3.4 Use cases of eBPF

eBPF has found applications in a wide range of domains. Some notable use cases include:

- Networking: eBPF is widely used to optimize network performance by enabling efficient packet filtering, load balancing, and traffic shaping. For example, tools such as Cilium [21] leverage eBPF to implement high-performance Kubernetes networking and security policies.
- Observability: eBPF provides deep visibility into system and application behavior without requiring invasive instrumentation. Tools such as BPF Compiler Collection (BCC) and bpftrace allow developers to trace system calls, monitor file I/O, and analyze performance bottlenecks in real time.
- Security: eBPF enables runtime security enforcement by monitoring system calls, file access, and network activity. It can detect and prevent malicious behavior, such as privilege escalation attempts or unauthorized data exfiltration.
- Tracing and Profiling: eBPF can be used to trace function calls, measure latency, and profile applications, making it invaluable for debugging and performance tuning.

3.4 Problem statement

Service meshes in distributed systems face significant inefficiencies in data transfer due to the prevalence of small, unaggregated packets. These inefficiencies manifest as such:

- High Latency: Frequent small-packet transmissions introduce delays from protocol overhead (e.g., TCP headers, ACKs) and kernel processing.
- Low Throughput: Low network bandwidth caused by excessive packet fragmentation and interrupt handling.
- Energy Overhead: Increased CPU cycles for per-packet processing, increasing power consumption in data centers.

Current buffering algorithms are static and do not adapt to dynamic network conditions (e.g., variable RTT, congestion) or application-specific requirements (e.g., latency-sensitive vs. batch traffic).

Research Gap: Lack of adaptive, context-aware buffering mechanisms capable of dynamically balancing these trade-offs based on real-time network state and traffic patterns.

4. The Dynamic Adaptive Packet Buffering (DAPB) algorithm

4.1 The DAPB algorithm's technical architecture

The DAPB algorithm is a novelty designed to improve data transfer efficiency in modern networking environments. It operates within 4 cornerstones (Fig. 3):

4.1.1 Data Collection and Learning Layer

The architecture begins with metric collectors in user space, which gather real-time network data (e.g., latency, throughput, packet loss) from the Linux network stack and service instances. These metrics feed into a reinforcement learning model that optimizes buffering policies through continuous interaction with the environment. Historical data are stored for history-based recommendations, while an optimization neural network, tuned via differential evolution, refines decision-making parameters. This layer ensures DAPB adapts dynamically to changing network conditions and application needs.

4.1.2 Decision and Control Plane

The decision model synthesizes inputs from the learning layer to generate adaptive buffering policies. It balances competing objectives (e.g., latency vs. throughput) using the reinforcement model's predictions. The control plane enforces these policies across the system, coordinating with the policy applier to translate decisions into actionable rules. This centralized intelligence allows DAPB to adjust buffer sizes, timeouts, and aggregation strategies in real time, tailored to specific traffic patterns (e.g., prioritizing VoIP packets over file transfers).

4.1.3 Kernel-Level Execution

Policies are executed in kernel space via eBPF data structures, which enable efficient packet processing without modifying the kernel. The eBPF components intercept traffic at the socket and TCP/IP layers, applying buffering rules while minimizing overhead. By operating close to the Network Interface Controller (NIC), DAPB reduces context switches and leverages kernel bypass techniques when possible. This design ensures low-latency processing while maintaining compatibility with existing Linux networking infrastructure.

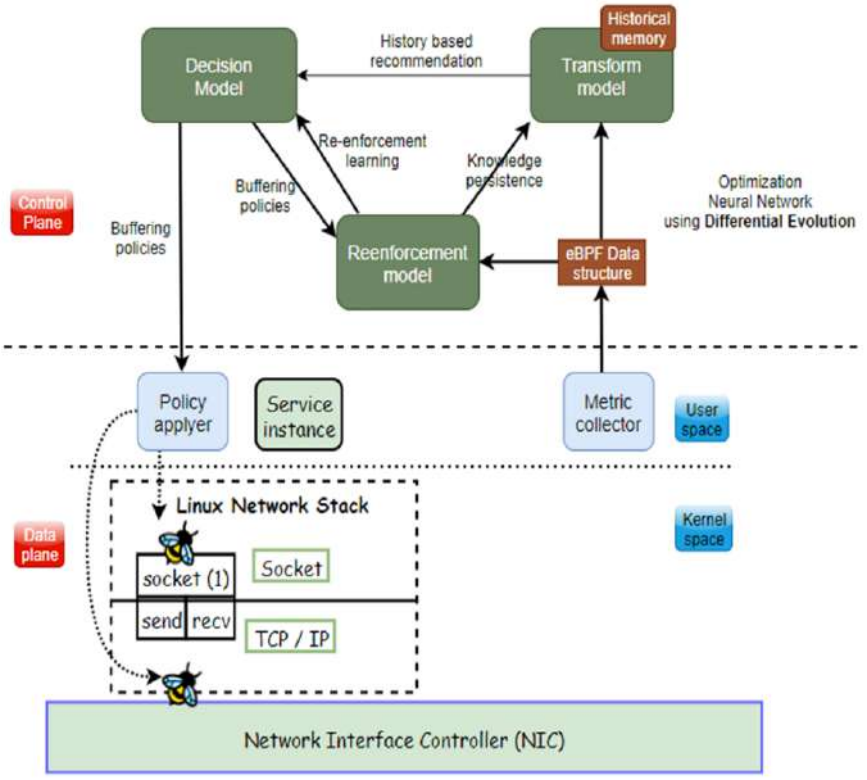


Fig. 3. The DAPB algorithm's technical architecture.

4.1.4 Feedback and Optimization Loop

The architecture closes the loop with knowledge persistence, where outcomes of applied policies (e.g., actual latency improvements) are logged and fed back into the learning layer. This continuous feedback enables the system to refine its models, ensuring long-term adaptability. The integration of differential evolution further optimizes neural network weights, while the control plane orchestrates iterative policy updates. Together, these components create a self-tuning system that evolves with network demands, achieving optimal performance across diverse service mesh environments.

As a result, the DAPB algorithm can be deployed and operate at the level of a whole service mesh installation (Fig. 4), managing buffering simultaneously for all containers within the installation.

4.2 The DAPB algorithm's features

The DAPB algorithm introduces several innovative features to address the limitations of existing buffering techniques.

4.2.1 Context-sensitive buffering

Unlike Nagle's algorithm, which uses a one-size-fits-all approach, DAPB tailors its buffering strategy to the specific needs of the application. For example, in latency-sensitive applications such as VoIP or online gaming, DAPB minimizes buffering to reduce delays. In contrast, in throughput-sensitive applications such as file transfers, it maximizes buffering to improve efficiency.

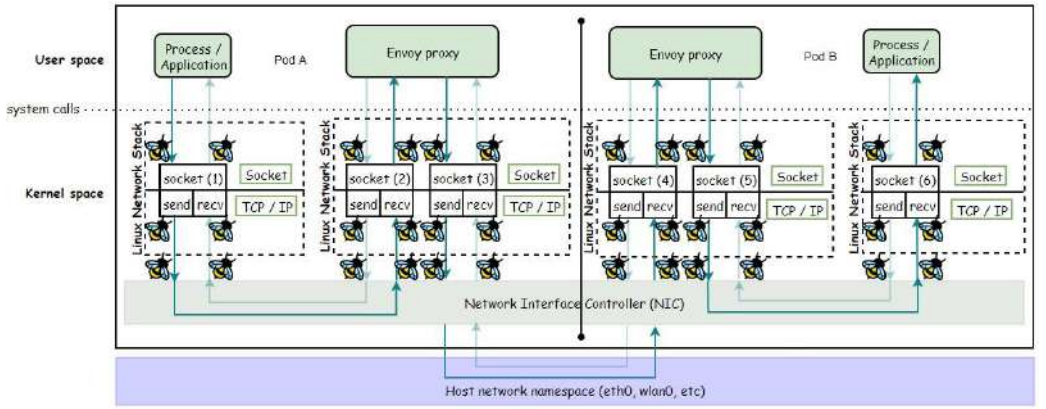


Fig. 4. Packet buffering using eBPF in Linux Kernel throughout service mesh.

Considering the following:

- $B(t)$ – the buffer size at time t, in bytes.
- $L(t)$ – the latency sensitivity of the application at time t (e.g., $L(t) = 1$ for latency-sensitive applications, $L(t) = 0$ for throughput-sensitive applications).
- $C(t)$ – the network conditions at time t, including round-trip time (RTT), packet loss rate ρ, jitter (J).

$$B(t) = \begin{cases} B_{min} & \text{if } L(t) = 1 \text{ (latency-sensitive)} \\ B_{max} & \text{if } L(t) = 0 \text{ (throughput-sensitive)} \\ f(C(t)) & \text{otherwise} \end{cases}, \quad (1)$$

where:

- B_{min} – the minimum buffer size for latency-sensitive applications (in bytes).
- B_{max} – the maximum buffer size for throughput-sensitive applications (in bytes).
- $f(C(t))$ – a function that adjusts the buffer size based on network conditions.

$$f(C(t)) = B_{min} + \alpha \cdot \text{RTT}(t) + \beta \cdot \rho(t) + \gamma \cdot J(t), \quad (2)$$

where α, β, *and* γ are weighting factors.

4.2.2 Adaptive timeout

While Nagle's algorithm relies on a fixed timeout, DAPB dynamically adjusts the waiting timeout based on real-time network conditions. If the network is congested, DAPB increases the timeout to allow more data buffering, thereby improving efficiency. However, if the network is underutilized, it reduces the timeout to minimize latency. This dynamic approach ensures that the DAPB strikes the right balance between efficiency and responsiveness. The timeout $T(t)$ is adjusted dynamically based on network conditions:

$$T(t) = T_{base} + \delta \cdot RTT(t) + \epsilon \cdot \rho(t) + \xi \cdot J(t), \quad (3)$$

where:

- $T(t)$ – the timeout period at time t (in ms).
- T_{base} – the base timeout value (in ms).
- δ, ϵ, *and* ξ – weighting factors that determine the influence of RTT, packet loss, and jitter on timeout.

4.2.3 Machine learning-driven predictions

DAPB incorporates machine learning-driven predictions to optimize its performance. The algorithm uses an AI model to predict network traffic. It provided a solid foundation for making informed decisions about buffer sizes and timeout values. For example, in a video streaming application, the AI component might analyze past transfer patterns to predict the best buffer size for a given video quality. This predictive capability ensures that DAPB remains effective even as network conditions and application requirements evolve. The predicted optimal buffer size $P(t)$ is derived from a machine learning model MM:

$$P(t) = M(C(t), H(t)), \tag{4}$$

where:

- $P(t)$ – predicted optimal buffer size at time t, derived from an AI model (in ms).
- $C(t)$ – current network conditions.
- $H(t)$ – historical data (e.g., past buffer sizes, network conditions, and performance metrics).

The buffer size $B(t)$ is then updated based on the prediction:

$$B(t) = min(B_{max}, max(B_{min}, P(t))). \tag{5}$$

4.2.4 Selective buffering

The algorithm intelligently decides which packets to buffer (and aggregate) and which to send immediately. Small packets that are part of a larger data stream are aggregated to reduce overhead, while urgent packets (e.g., control messages) are sent immediately to minimize latency. This selective approach ensures that the DAPB maintains high performance without compromising responsiveness. The average urgency of packets in the buffer at time t is given by:

$$U(t) = \frac{1}{N(t)} \sum_{i=1}^{N(t)} u_i(t), \tag{6}$$

where:

- $u_i(t)$ is the urgency of the i-th packet at time t ($0 \le u_i(t) \le 1$),
- $U(t)$ is the average urgency at time t,
- $N(t)$ is the number of packets in the buffer at time t,

The urgency $u_i(t)$ can be established either by using protocol headers or the application context. The first approach consists of extracting priority flags (e.g., HTTP/2 stream priorities, DSCP/ToS bits in IP headers, or gRPC metadata). The second consists of integrating with service mesh APIs (e.g., Istio virtual service) to label latency-sensitive traffic (e.g., VoIP, gaming) as high urgency. The decision to send the buffer is based on the following condition: send buffer if $U(t) > U_{th}$ or $B(t) \ge B_{max}$, where U_{th} is a threshold for the urgency of the packet.

4.2.5 Energy efficiency

Energy efficiency is also a priority for DAPB. By optimizing buffer sizes and timeout values, the algorithm reduces unnecessary resource consumption, making it particularly valuable for energy constrained environments like edge computing and IoT devices. For example, in an IoT sensor network, DAPB can minimize buffer sizes during periods of low activity, saving energy without compromising performance. The energy cost $E(t)$ is modeled as:

$$E(t) = \eta \cdot B(t) + \theta \cdot T(t), \tag{7}$$

where η and θ are weighting factors that represent the energy cost of buffering and waiting, respectively.

5. Performance Metrics

After applying the DAPB algorithm to service mesh using eBPF in Linux, 6 performance metrics should be collected (evaluated) and analyzed. Each of them is presented below.

5.1 Reduction in small packets

This metric quantifies the decrease in the number of small data packets transmitted over the network due to buffering. By holding data in a buffer until a predefined size or timeout is reached, fewer packets are sent, reducing overhead and eliminating network congestion. For example, if an application generates 1000 small packets per second but transmits only 200 after aggregation, 800 packets are eliminated, lowering processing demands on network hardware. Considering the following variables:

- N_{reduced} – the reduction in small packets (in bytes).
- $\lambda(t)$ – the arrival rate of the packets at time t (in packets/sec).
- $N_{\text{DAPB}}(t)$ – the number of packets transmitted after buffering, at time t (in bytes).

$$N_{\text{reduced}} = \sum_{t=0}^{T} (\lambda(t) - N_{\text{DAPB}}(t)) \tag{8}$$

5.2 Buffer efficiency

Evaluates how effectively the allocated buffer capacity is used to aggregate the data. High efficiency means that the buffer is consistently filled to its maximum capacity before transmission, minimizing wasted space. Lower efficiency indicates frequent early transmissions (e.g. due to timeouts), which may under-utilize buffer resources and reduce potential throughput gains. This is important because high efficiency directly means a reduction in latency (fewer waiting for partial fills). Considering the following:

- $B_{\text{max}}(t)$ – the maximum buffer size at time t (in bytes).
- $B(t)$ – the actual data accumulated in the buffer at time t (in bytes).
- $Tr_{\text{DAPB}}(t)$ – the transmissions triggered by buffer-full or timeout events (in ms).
- η – the buffer efficiency.

$$\eta = \frac{\sum_{t=0}^{T} (B_{\text{used}}(t) \cdot Tr_{\text{DAPB}}(t)}{\sum_{t=0}^{T} B_{\text{max}}(t)} \tag{9}$$

5.3 End-to-end latency

Estimates the time it takes for a request to traverse all nodes in the service mesh, including both network delays and buffering pauses. Highlights the bottlenecks where buffering dominates latency, guiding changes such as adjusting buffer sizes or timeouts to maintain responsiveness across distributed services. For each node v, considering the following:

- Q_v as a queue with capacity B_{max}.
- Edges e_{uv} have a transmission delay d_{uv}.

The Nagle-inspired policy modifies the dequeue behavior of Q_v. Packets are dequeued only when $\|Q_v\| \geq B_{\text{max}}$, or τ expires. After applying the DAPB algorithm, for a path $P = (v_1, v_2, \ldots, v_n)$, the end-to-end latency becomes:

$$L_P = \sum_{i=1}^{n-1} \left(d_{v_i, v_{i+1}} + \Pi_{Q_{v_i} < B_{\max}} \cdot \tau \right), \quad (10)$$

where:

- $d_{vi,vi+1}$ — the network delay between nodes v_i and v_{i+1} (in ms).
- Π — an indicator function for delayed transmissions (in ms).
- $\Pi_{Qvi<Bmax} \cdot \tau$ — the buffering delay at node v_i if its buffer is not yet full (in ms).

5.4 Additional delay

This metric estimates the additional delay introduced by buffering packets before transmission. While aggregation improves throughput, it inherently adds waiting time, either until the buffer fills or a timer expires. Applications sensitive to delays (e.g., real-time systems) must balance this trade-off carefully to avoid degrading user experience. Considering the following variables:

- τ – the acknowledgment timeout (in ms).
- t_{fill} – the time to fill the buffer to $B_{\max}$ (in ms).
- L_{added} – the added latency increase (in ms).

$$L_{added} = \mathbb{E}[\min(\tau, t_{\text{fill}})] \quad (11)$$

The increase in latency can be useful to estimate the effect of buffering on latency-sensitive applications (e.g., real-time APIs). For example, if $t_{\text{fill}} = 150ms$ and $\tau = 200ms$, the added latency is 150 ms. However, if $t_{\text{fill}} = 250ms$, the added latency is 200 ms (the timeout triggers transmission).

5.5 Throughput gain

The throughput gain reflects the improvement in data transmission rates achieved by sending larger aggregated packets instead of smaller ones. Larger packets reduce header overhead and improve network utilization, enabling more efficient bandwidth use. For example, combining 100 small packets into one large packet minimizes repetitive header transmissions, increasing throughput.

5.6 eBPF overhead cost

While eBPF optimizes kernel-level processing, its operations consume additional CPU cycles. This metric assesses the computational cost of using eBPF to manage packet aggregation. The cost must remain low enough to avoid negating the benefits of aggregation, ensuring net performance gains. It can be estimated as the additional processing time, memory increase, and energy consumption introduced by eBPF hooks to intercept, buffer, and redirect packets. Considering the following variables:

- $X^{\text{proc}}_{\text{native}}$ – the time (or memory or energy) to process a packet in the native Linux stack.
- $X^{\text{proc}}_{\text{eBPF}}$ – the time (or memory or energy) of all eBPF logics (e.g., aggregation, buffering).
- C_{eBPF} – the overhead cost induced by eBPF.

$$C_{eBPF} = X^{\text{proc}}_{\text{eBPF}} - X^{\text{proc}}_{\text{native}} \quad (13)$$

An important aspect of this indicator is that it allows to verify that eBPF enhancements are not less than the cost due to increased resource usage. For example, if C_{eBPF} = 5 μs/packet, and the packet arrival rate λ(t) = 1000 packets/sec the overhead is about 5 seconds of CPU time per second.

5.7 Performance metrics conclusion

The metrics can be interpreted holistically as follows:

- Reduction in Small Packets (N_{reduced}) and Throughput Gain (T_{gain}) measure improvements in network efficiency from aggregation. These are critical for throughput-sensitive applications (e.g., file transfers).
- End-to-End Latency (L_P) and Latency Increase (L_{added}) capture responsiveness trade-offs, vital for real-time systems (e.g., VoIP).
- The buffer efficiency (η) reflects resource utilization, indicating how well DAPB adapts buffer usage to dynamic conditions.

The context-specific guidance for the ML models is as follows:

- In throughput-sensitive context: prioritize N_{reduced}, T_{gain}, and η. The cost (C_{eBPF}) is tolerable if the gains exceed it.
- In latency-sensitive context: minimize L_P and L_{added}; tolerate lower η or higher C_{eBPF}.
- In energy-constrained context (Edge/IoT, etc.): favor η and low C_{eBPF}.

6. Risks and limitations

The DAPB algorithm introduces significant improvements over static buffering approaches, but its adaptive and machine learning-driven nature presents several challenges that must be carefully mitigated.

1. Prediction Inaccuracies in Dynamic Environments

   Risk: the machine learning model's reliance on historical data and real-time metrics may yield suboptimal predictions under sudden network changes (e.g., flash crowds, DDoS attacks). Noisy or incomplete data (e.g., inaccurate RTT measurements due to asymmetric routes) could degrade performance.

   Mitigation: incorporate ensemble methods (e.g., random forests [23]) to reduce variance and fallback mechanisms (e.g., reverting to Nagle-like static thresholds when prediction confidence is low).

2. Overhead from Adaptive Mechanisms

   Risk: the computational cost of dynamically adjusting buffer sizes and timeouts may offset throughput gains, especially in resource-constrained edge/IoT environments. The eBPF overhead metric must be monitored to ensure net benefits.

   Mitigation: profile the algorithm's CPU/memory footprint under varying loads and optimize the eBPF bytecode (e.g., reducing redundant calculations in the $f(C(t))$ and $T(t)$ functions).

3. Misclassification of Application Context

   Risk: incorrectly labeling an application as latency-sensitive ($L(t) = 1$) or throughput-sensitive ($L(t) = 0$) could lead to inappropriate buffering. For example, misclassifying VoIP traffic as batch processing would introduce unacceptable delays.

   Mitigation: implement hybrid labeling (e.g., allow applications to declare their sensitivity via API) and validate classifications using runtime telemetry (e.g., packet inter-arrival times).

4. Energy Trade-offs in Adaptive Buffering

   Risk: although DAPB optimizes energy use, frequent buffer resizing $B(t)$ and timeout adjustments $T(t)$ may increase CPU cycles, negating energy savings in low-power devices.

Mitigation: introduce hysteresis in adjustments (e.g., change $B(t)$ only when network conditions $C(t)$ shift beyond a threshold) to reduce computational churn.

5. Scalability in Large-Scale Deployments

   Risk: the centralized control plane in service meshes may struggle to propagate real-time network conditions $C(t)$ to all proxies, causing inconsistent buffering decisions across nodes.

   Mitigation: decentralize partial decision-making (e.g., let each proxy compute B(t) locally) and use lightweight consensus protocols for critical updates [24].

6. Security Implications of eBPF Dependencies

   Risk: eBPF's kernel-level access exposes DAPB to potential exploits (e.g., buffer overflow in eBPF programs). Maliciously crafted packets could trigger excessive buffering, leading to resource exhaustion.

   Mitigation: apply eBPF hardening techniques (e.g., verifier-based bounds checking, ratelimiting buffer allocations) and audit the DAPB eBPF code with tools like BPFKit [25].

7. Interoperability with Legacy Systems

   Risk: older kernels or non-Linux environments may lack eBPF support, limiting DAPB's applicability. Hybrid deployments (e.g., partial service meshes) could experience performance asymmetry.

   Mitigation: provide a fallback mode using socket-level buffering (e.g., TCP_CORK) with reduced adaptability, and document compatibility matrices.

7. Next steps of the research

1. Implement the new algorithm using eBPF.
2. Engineer the ML model.
3. Prepare multiple testing environments. Make sure to have most common architectures:
   - 2 containers inside a pod with one Istio sidecar (Intra-pod)
   - 2 containers inside 2 pods with 2 Istio sidecars (Inter-pod)
   - n containers inside n pods with n Istio sidecars (Inter-pod), where $2 < n < \infty$.

   There are two environments for each architecture. The new algorithm is applied to the first one, and it is not applied to the second one.
4. Collect basic system metrics. They include, but are not limited to, the CPU, the memory (RAM), and the disk usage (in bytes).
5. Integrate the new algorithm into the corresponding testing environments.
6. Collect and store performance metrics. These are the metrics described in section 5.
7. Analyze and assess the baseline system and performance metrics.
8. Conclude the work.

8. Conclusion

This paper introduces the Dynamic Adaptive Packet Buffering (DAPB) algorithm. It is designed to enhance data transfer efficiency in service mesh environments by leveraging eBPF. DAPB improves upon existing buffering algorithms like Nagle's algorithm by dynamically adjusting buffer sizes and timeout values based on real-time network conditions, application requirements, and machine learning predictions. Key features include context-sensitive buffering, adaptive timeout mechanisms, selective aggregation, and energy efficiency optimizations, making it suitable for diverse scenarios such as latency-sensitive applications and resource-constrained IoT devices.

Performance metrics can be used to assess the reduction in small packets, the improved throughput, and the added latency

References

[1]. J. Nagle, Congestion control in IP/TCP internetworks, RFC Editor, RFC 896, Jan. 1984, Obsoleted by RFC 1122, but foundational to Nagle's algorithm. [Online]. Available: https: //tools.ietf.org/html/rfc896.

[2]. J. Nagle, "Congestion control in IP/TCP internetworks," RFC Editor, RFC 896, (Jan. 1984), Obsoleted by RFC 1122, but foundational to Nagle's algorithm. [Online]. Available: https: //tools.ietf.org/html/rfc896.

[3]. TCP/IP Illustrated, Volume 1: The Protocols ([1994]), W. R. Stevens, Addison-Wesley Professional, isbn: 978-0201633467.

[4]. "The BSD packet filter: a new architecture for user-level packet capture" ([1993]), S. McCanne et al., (In: Proceedings of the USENIX Winter 1993 Conference), pp. 259–270.

[5]. "BPF: In-kernel Virtual Machine" ([2015]), A. Starovoitov, (In: Linux Plumbers Conference).

[6]. "The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel" ([2018]), T. Hoiland-Jorgensen et al., (In: Proceedings of the 14th International Conference on Emerging Networking Experiments and Technologies), pp. 54–66, doi: 10.1145/3281411.3281443.

[7]. Computer Networks ([2011]), A. S. Tanenbaum et al., Pearson.

[8]. "BBR: Congestion-based congestion control" ([2016]), N. Cardwell, Y. Cheng, C. S. Gunn, et al., ACM Queue, 14, 5, pp. 20–53, doi: 10.1145/3012426.3022184.

[9]. N. Cardwell, Y. Cheng, S. H. Yeganeh, et al., "Tcp bbr v2 alpha/release history," IETF, RFC 8962, (2021). [Online]. Available: https://tools.ietf.org/html/rfc8962.

[10]. Understanding Linux Network Internals ([2005]), C. Benvenuti, O'Reilly, isbn: 9780596002558.

[11]. "Efficient data transfer through zero copy" ([2006]), W. Ma et al., (In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing), pp. 1–12, doi: 10.1145/1188455.1188583.

[12]. "Differential evolution optimization for constrained routing in Wireless Mesh Networks" ([2014]), M. Sanni et al., (In: International Conference on Frontiers of Communications, Networks and Applications (ICFCNA 2014 - Malaysia)), pp. 1–6, doi: 10.1049/cp.2014.1397.

[13]. Istio, Istio: A service mesh for microservices, Official documentation, (2023). [Online]. Available: https://istio.io/latest/docs/concepts/what-is-istio/.

[14]. "Congestion control in IP/TCP internetworks" ([1984]), J. Nagle, ACM SIGCOMM Computer Communication Review, 14, 4, pp. 11–17, doi: 10.1145/1024908.1024910.

[15]. "Congestion avoidance and control" ([1988]), V. Jacobson, ACM SIGCOMM Computer Communication Review, 18, 4, pp. 314–329, doi: 10.1145/52325.52341.

[16]. R. Braden, "Requirements for internet hosts—communication layers," IETF, RFC 1122, (1989). [Online]. Available: https://tools.ietf.org/html/rfc1122.

[17]. "Reducing web latency: the virtue of gentle aggression" ([2013]), T. Flach et al., (In: Proceedings of the ACM SIGCOMM 2013 Conference), pp. 159–170, doi: 10.1145/2486001.2486030.

[18]. "Evaluating the impacts of alternative TCP congestion control algorithms" ([2008]), S. Ha et al., (In: IEEE International Conference on Network Protocols), pp. 49–58, doi: 10.1109/ICNP. 2008.4697036.

[19]. "TCP Vegas: End to end congestion avoidance on a global internet" ([1995]), L. S. Brakmo et al., IEEE Journal on Selected Areas in Communications, 13, 8, pp. 1465–1480, doi: 10.1109/ 49.464716.

[20]. Computer Networking: A Top-Down Approach ([2021]), J. F. Kurose et al., Pearson, isbn: 9780135928615.

[21]. Learning eBPF ([Mar. 2023]), L. Rice, O'Reilly, isbn: 978-1-098-13887-5.

[22]. T. Graf et al., "Cilium: eBPF-based networking, security, and observability," Isovalent, Tech. Rep., (2023), Official documentation. [Online]. Available: https://docs.cilium.io/en/stable/index.html.

[23]. "Network Shortcut in Data Plane of Service Mesh with eBPF" ([Jan. 2024]), W. Yang et al., Journal of Network and Computer Applications, 222, 1, p. 103805, doi: 10.1016/j.jnca. 2023.103805.

[24]. A. Cutler et al., "Random forests," in Research Gate, (Jan. 2011), vol. 45, pp. 157–176, isbn:978-1-4419-9325-0. doi: 10.1007/978-1-4419-9326-7_5.

[25]. "Reaching Consensus in the Byzantine Empire: A Comprehensive Review of BFT Consensus Algorithms" ([Jan. 2024]), G. Zhang et al., ACM Comput. Surv., 56, 5, doi: 10.1145/3636553.

[26]. Gui774ume, eBPFKit: A rootkit and intrusion detection system based on ebpf, https:// github.com/Gui774ume/ebpfkit, GitHub repository, (2021). [Online]. Available: https://github.com/Gui774ume/ebpfkit.

Информация об авторах / Information about authors

Ханк-Дебэн ДЖАМБОНГ ТЕНКЕ – магистр программной инженерии, аспирант НИУ “Высшая Школа Экономики”, приглашенный преподаватель на факультет компьютерных наук НИУ “Высшая Школа Экономики”. Сфера научных интересов: инженерия программного обеспечения и компьютерных систем.

Hank-Debain DJAMBONG TENKEU – Master of Science in Software Engineering, postgraduate student at the National Research University “Higher School of Economics”, invited lecturer at the Faculty of Computer Science of NRU “Higher School of Economics”. Research interests: Software and Computer Systems Engineering.

Дмитрий Владимирович АЛЕКСАНДРОВ является Профессором в департаменте программной инженерии факультета компьютерных наук у НИУ “Высшая Школа Экономики”. Он также является заведующим научно-учебной лаборатории облачных и мобильных технологий. Его научные интересы включают методы и технологии искусственного интеллекта, машинное обучение и анализ данных, iOS разработка, разработка мобильных приложений, разработка программного обеспечения, indoor-навигация, базы данных, разработка игр.

Dmitry Vladimirovich ALEXANDROV is a Professor in the Department of Software Engineering, Faculty of Computer Science, National Research University “Higher School of Economics”. He is also the Head of the Research and Educational Laboratory of Cloud and Mobile Technologies. His research interests include methods and technologies of artificial intelligence, machine learning and data analysis, iOS development, mobile application development, software development, indoor navigation, databases, game development.

DOI: 10.15514/ISPRAS-2025-37(5)-8

Tuning LLM in Secure Code Generation

[1,2,3] *D.S. Shaikhelislamov, ORCID: 0000-0002-9734-7937 <shaykhelislamov.ds@ispras.ru>*
[4] *M.S. Varetsa, ORCID: 0009-0003-8837-5252 <varetsa.m.s@nanosemantics.ai>*
[3] *A.S. Syomkin, ORCID: 0009-0004-3388-7282 <assemkin@edu.hse.ru>*
[5] *O.Yu. Rogov, ORCID: 0000-0001-9672-2427 <rogov@airi.net>*

[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, 9, Institutsky lane, Dolgoprudny, Moscow region, 141700, Russia.*

[3] *National Research University, Higher School of Economics, 20, Myasnitskaya ulitsa, Moscow, 101978, Russia.*

[4] *Russian Technological University MIREA, 78, Vernadsky Ave, Moscow, MIREA, Russia.*

[5] *AIRI, 32k1, Kutuzovsky ave., Moscow, 121170, Russia.*

Abstract. The popularity of using LLM for code generation makes it mandatory to comprehensively verify the security and reliability of the generated code. To verify the generated code, it is suggested to use the static analyzer Svace, which checks the executable code using the built-in compiler and checks the code for weaknesses. The result of the generation is processed using Svace and receives prompts with detected warnings or errors in the code and requests corrections from LLM after generation. In addition, we fine-tune the Qwen2.5-Coder model using direct preference optimization (DPO) for error code pairs that include common syntax errors and runtime errors. This reduced the error rate, including syntactic errors and vulnerabilities, by 20\%. To evaluate the models, we collected a specialized dataset from open sets for LLM evaluation, focusing on tasks in which the models generate erroneous code. The experimental results show that fine-tuning the model with a focus on code quality allows you to generate code that reduces typical errors. In this work, we combine an iterative prompting mechanism with DPO to improve the security and accuracy of LLM code generation.

Keywords: code generation; large language models; static analysis; analyzer feedback; code security; fine-tuning.

For citation: Shaikhelislamov D.S., Varetsa M.S., Syomkin A.S., Rogov O.Yu. Tuning LLM in secure code generation. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 111-122. DOI: 10.15514/ISPRAS-2025-37(5)-8.

Настройка языковой модели для безопасной генерации кода

[1,2,3] *Д.С. Шайхелисламов, ORCID: 0000-0002-9734-7937 <shaykhelislamov.ds@ispras.ru>*
[4] *М.С. Вареца, ORCID: 0009-0003-8837-5252 <varetsa.m.s@nanosemantics.ai>*
[3] *А.С. Сёмкин, ORCID: 0009-0004-3388-7282 <assemkin@edu.hse.ru>*
[5] *О.Ю. Рогов, ORCID: 0000-0001-9672-2427 <rogov@airi.net>*

[1] *Институт системного программирования им. В.П. Иванникова РАН,*
Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.

[2] *Московский физико-технический институт,*
Россия, 141700 Московская область, г. Долгопрудный, Институтский переулок, 9.

[3] *НИУ Высшая школа экономики,*
Россия, 101000, г. Москва, ул. Мясницкая, д. 20.

[4] *Российский технологический университет МИРЭА,*
Россия, 119454 г. Москва, проспект Вернадского, дом 78.

[5] *Институт искусственного интеллекта AIRI,*
Россия, 121170, г. Москва, Кутузовский проспект, д. 32 к. 1.

Аннотация. Популярность использования LLM для генерации кода делает обязательной всестороннюю проверку безопасности и надежности сгенерированного кода. Для проверки сгенерированного кода предлагается использовать статический анализатор Svace, который проверяет исполняемый код с помощью встроенного компилятора и проверяет код на наличие дефектов. Результат генерации обрабатывается с помощью Svace и получает запросы с обнаруженными предупреждениями или ошибками в коде и запрашивает исправления у LLM после генерации. Кроме того, настраиваем модель Qwen2.5-Coder, используя прямую оптимизацию предпочтений (DPO) для пар кодов ошибок, которые включают распространенные синтаксические ошибки и ошибки во время выполнения. Это снизило частоту ошибок, включая синтаксические и уязвимые места, на 20%. Для оценки моделей мы собрали специализированный набор данных из открытых наборов для оценки LLM, сосредоточив внимание на задачах, в которых модели генерируют ошибочный код. Результаты экспериментов показывают, что тонкая настройка модели с акцентом на качество кода позволяет генерировать код, который уменьшает количество типичных ошибок. В этой работе мы объединяем механизм итеративных запросов с DPO для повышения безопасности и точности генерации кода LLM.

Ключевые слова: генерация кода; большие языковые модели; статический анализ; обратная связь от анализаторов; безопасность кода; настройка моделей.

Для цитирования: Шайхелисламов Д.С., Вареца М.С., Сёмкин А.С., Рогов О.Ю. Настройка языковой модели для безопасной генерации кода. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 111–122 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)–8.

1. Introduction

In the modern world, large language models (LLMs) are simplifying the process of writing code and developing software. According to information from Google's CEO as of October 2024, AI generates approximately 25% of the code in Google's products [1]. The efficiency with which AI-based solutions generate code has encouraged users and developers of varying skill levels and experience to use these tools for quick problem solving in programming or to integrate AI-generated code into software systems and applications. However, in most cases, the results are not subject to any quality control, raising concerns about maintaining the security of IT product development processes. Authors [2] noted that AI-assistant for coding may recommend syntactically incorrect code including variables, functions, and attributes that are undefined or outside the scope of the codebase.

Training data may contain outdated functions and libraries, which can lead to vulnerabilities when used, and may also intentionally include erroneous or unsafe code used to poison large language

models during the training phase [3]. Recent research [4-5] confirms that approximately 40% of the code generated by large language models contains vulnerabilities, while user studies indicate that developers miss at least 10% of the bugs in AI-generated code. Vulnerabilities in software, often due to common programming errors, remain a primary attack vector for malicious actors, leading to significant financial losses, data breaches, and reputational risk [6]. The increasing complexity of software systems and the growing prevalence of cyber security threats have underscored the critical need for secure coding practices and preparation of secure datasets.

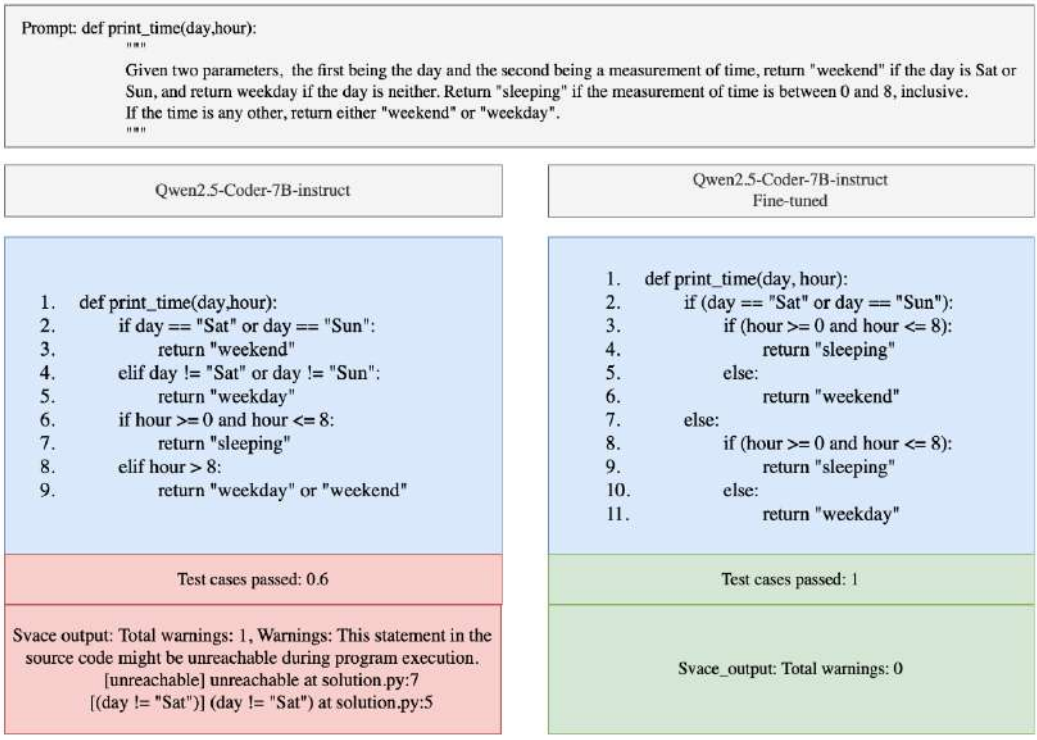


Fig. 1. Comparison of the source and modified code generated using LLM with the warning analysis of the Svace static analyzer.

To address this challenge, we are improving the CodePatchLLM [7], enriching a fine-tuned model that has finetuned on CodePreference dataset [8]. We emphasize secure coding patterns, enabling the model to learn not only syntactic and functional correctness but also robust defensive programming techniques.

Our work yields several findings:

- **Novel evaluation dataset**: We introduce the **MultiEval** dataset, designed to bridge the gap between functional code generation and security-aware programming. This dataset focuses on coding tasks that historically led to errors in LLM-generated code, providing a robust benchmark for evaluating model performance.
- **Fine-tuned model**: We enhance the Qwen2.5-Coder-7B-instruct model using direct preference optimization (DPO) [9], fine-tuning it on pairs of erroneous and correct code. This approach reduces both syntactic and runtime errors, resulting in a more reliable model for code generation.

2. Related work

LMs for Code Generation. Large LMs designed for general-purpose applications [10], exhibit the capability to generate functionally correct code [7, 11]. In [12], the authors analyze common vulnerabilities (for example, injections or buffer overflows) that occur when using LLM, and propose methods for detecting them using static analysis. This profound understanding of code is obtained through pretraining on extensive code corpora. More recently, synthetic coding-specific instructions have been employed to fine-tune pretrained LMs to further enhance their capabilities in functional correctness [13].

Program Security. An important aspect of programs is their security. Svace is an industry-leading static analysis engine for detecting security vulnerabilities [14]. It supports mainstream languages and provides queries for common CWEs. Recently, Svace has been a popular and reliable choice for evaluating the security of LM-generated code [15]. It is also presented as the main element of the prompt tuning pipeline with LM in the CodePathLLM framework.

Authors in [16] use expensive manual inspection to curate their training dataset. In contrast, our work leverages an automated data collection pipeline with SAST, resulting in a diverse dataset with broader coverage of CWEs and programming languages.

Security of LM-generated Code. Several studies have assessed the security of code generated by pretrained LMs. These investigations highlight a common finding: all evaluated LMs frequently produce security vulnerabilities. Addressing this significant security concern is still an early-stage research topic. The seminal works of SVEN [16] and SafeCoder [13] offer two different approaches: instruction tuning and fine-tuning the LM. CodePatchLLM [5] combines both approaches. Fine-tuning LLM to improve code quality is explored in [17], which shows that training on specialized datasets with examples of secure patterns increases the reliability of generation. In [5], an approach was proposed to integrate static analyzers such as Svace into the generation process for automatic code verification at the inference stage.

3. Background and Problem Statement

In this section, we present the necessary background knowledge and outline the problem setting.

3.1 Instruction tuning with Svace

More information about how the instructional process works can be found in early works [7]. The whole process can be broken down into three key steps: (1) code generation according to a given description; (2) code verification by the Svace static analyzer; (3) instruction enrichment with messages from Svace. Automatic correction is performed sequentially with feedback steps until the stop condition is met. The condition for stopping is either reaching the limit of iteration t_{max}, or until all defects in the generated code are fixed. We illustrate this mechanism in Fig. 2. The LMs are fine-tuned to follow task-specific instructions and align with human preferences – security.

3.2 Fine-tuning LM

We employed a fine-tuning method for LM that generate code, aiming to enhance the quality and safety of the generated code. For the fine-tuning process, we adopted a reinforcement learning method through Direct Preference Optimization (DPO). The key idea is to use pairwise comparison data when a preference is indicated between two model outputs with the same input data. Given a dataset $D = \{(x_i, y_i^-, y_i^+)\}_{i=1}^N$ where x_i is the input, y_i^+ is the preferred output, and y_i^- is the less preferred output, DPO aims to maximize the likelihood of the preferred outputs while minimizing the likelihood of the less preferred ones. The objective function for DPO can be written as [29]:

$$L_{DPO}(\pi_\theta, \pi_{ref}) = -E_{(x,y^+,y^-)} \sim D\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y^+|x)}{\pi_{ref}(y^+|x)} - \beta\log\frac{\pi_\theta(y^-|x)}{\pi_{ref}(y^-|x)}\right)\right],$$

where:

- π_θ is the policy (model) being optimized,
- π_{ref} is a reference policy (usually the pre-trained model),
- σ is the sigmoid function,
- β is a hyperparameter controlling the strength of the preference signal.

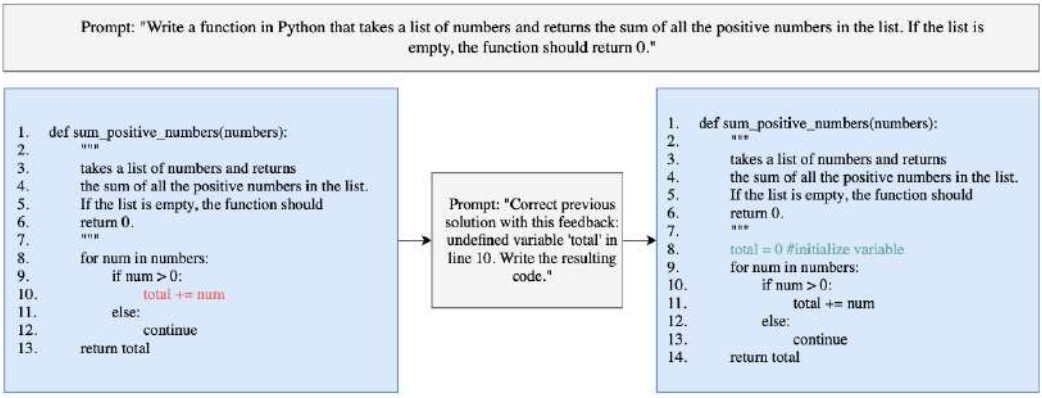


Fig. 2. An example of correcting an error in the code generated using LLM: initializing a variable for the correct execution of a function.

This objective encourages the model to assign higher probabilities to preferred outputs y_i^+ relative to the less preferred outputs y_i^-, while staying close to the reference policy π_{ref} to prevent overfitting. Unlike RLHF, which involves training a reward model and then using reinforcement learning to optimize the policy, DPO directly optimizes the policy using a simple classification objective. This makes DPO more computationally efficient and easier to implement.

Our goal is to address the limitation of existing LMs infrequently producing unsafe code, as highlighted in Fig. 1 (left). While improving security is critical, it is equally important for the enhanced LMs to achieve high utility, such as generating functionally correct code or solving natural language tasks. Therefore, our dual objective involves simultaneously improving security and utility. To achieve this goal, we focus on both methods: fine tuning model and tuning instructions.

4. Experiments

In this section, we outline the experimental setup for our study evaluating the safety and reliability of code generated by large language models (LLMs). Our experiments are conducted using the framework BigCodeEval [18]. We aim to determine whether an iterative feedback mechanism (framework CodePatchLLM [7]) with a fine-tuned model can significantly improve the accuracy and reliability of code generation. Additionally, we explore the impact of DPO on enhancing the Qwen2.5-Coder-7B-instruct [19] model performance in generating error-free code. To ensure the reproducibility of results, the LLM's temperature was set to 0 in all experiments unless otherwise specified. This parameter configuration minimizes random variation in the model's outputs, thereby enhancing the reliability of the findings.

4.1 Tasks & Datasets

In the course of our comprehensive study, we performed a detailed comparison of the models in the context of a Python code generation task. To facilitate this evaluation, our primary benchmark is HumanEval [20], a popular dataset for assessing the performance of code generation models.

Additionally, we developed and implemented a distinctive dataset MultiEval specifically designed to evaluate the quality of code generated by large language models (LLMs) using data that are representative of programming scenarios.

MultiEval is a set of tasks selected from open-source datasets to evaluate code-generating models. To construct this dataset, we drew upon several publicly available task sets aimed at evaluating the quality of generative code models. Among these, we focused on datasets such as APPS-Interview and APPS-Introductory [21], StudentEval [22], Mercury [23], CoNaLa [24], MBPP [25], DS-1000 [26]. Total 16 534 NL-Code tasks that are popular for LLM skills research. Each of these datasets provides a diverse array of tasks that encompass a wide range of programming concepts and practices.

For each task, a solution was generated by a model from the Qwen family: Qwen2.5-Coder-7B, Qwen2.5-Coder-3B, Qwen2.5-Coder-1.5B (in regular and instruct versions), as well as Qwen2.5-3B, Qwen2.5-7B and Qwen2.5-14B. The criterion for including the task in the final set was the presence of errors in the generated solution on the first attempt, determined using the Svace static analyzer. As a result, 376 tasks were selected, forming the final data set.

The quality metric is calculated as the ratio of the number of tasks solved without syntactic or logical errors to the total number of tasks in the dataset. This approach allows an objective assessment of the model's ability to generate correct code the first time.

4.2 Metrics

The primary quality metric was the proportion of problems solved without errors, calculated as follows:

$$ErrorFree\ Rate = \frac{N_{error-free}}{N_{total}} * 100\%,$$

where $N_{error-free}$ is the number of error-free solutions, and N_{total} is the total number of tasks.

Here, an error-free solution is defined as code that passes all static analysis checks performed by Svace without any critical issues. For this metric, we determined the percentage of tasks resolved without errors on the first generation. This metric is reported for both the HumanEval and MultiEval datasets, providing a comprehensive comparison of model performance across different task complexities and domains.

When evaluating on the HumanEval dataset, we employed an additional metric: pass@1. This metric measures the likelihood that a model generates a correct solution on its first attempt. The pass@1 score was calculated using the unit tests provided in the original dataset, as defined by the following formula:

$$pass@1 = \frac{N_{correct}}{N_{total}} * 100\%,$$

where $N_{correct}$ is the number of correct solutions on the first attempt, and N_{total} is the total number of tasks.

A solution was considered correct upon the first generation if the generated code passed all unit tests for the given task. This metric is particularly useful for assessing the model's ability to produce accurate and functional code.

4.3 Evaluation of fine-tuned model

The CodePreference dataset [27] was chosen as the basis for fine tuning, which consists of a set of tasks accompanied by prompts and code pairs. These code pairs include both correct and incorrect code, reflecting real scenarios that developers encounter during programming. The selection of the CodePreference dataset was driven by several factors. Firstly, it provides a variety of scenarios, ensuring the testing of the model in conditions that closely resemble situations with using LLM for coding. Furthermore, the richness of error types within the code enables our model to learn not only to generate syntactically correct code but also to detect and correct potential mistakes.

We also tested the DPO model fine-tuning method on another dataset in the context of improving overall code security. To achieve this goal, the CVEFixes dataset [28] was selected. CVEfixes is a comprehensive vulnerability database that is automatically collected and curated from Common Vulnerabilities and Exposures (CVE). This dataset contains examples of vulnerable code for various languages (C, Python, Java, etc.) and is presented in sqlite database format. We combined the strings from this database and compiled a dataset in jsonl format consisting of 45748 pairs.

The retraining process for the Qwen2.5-Coder-7B-instruct model was carried out in three iterations. In each iteration, we utilized data from the CodePreference dataset to train the model, embedding an algorithm that allows it to adapt to the received data based on feedback. Throughout each iteration, the model improved its capabilities by learning from the errors identified in previous versions.

Each iteration included the analysis of results, enabling the tracking of progress and adjustments in the training process. As a result, we obtained a fine-tuned Qwen2.5-Coder-7B-instruct model, which demonstrated a significant enhancement in code quality, as well as an ability to effectively identify and correct common errors.

To further analyze the performance of the models, we compared the results of the fine-tuned Qwen2.5-Coder-7B-instruct model with its original version. The resulting metrics, including the error-free rate and pass@1 scores, are presented in Table 1. These results highlight the effectiveness of fine-tuning in enhancing the model's code generation capabilities.

Table 1. Error-Free Rate (EFR) and pass@1 metric for fine-tuned and original models on HumanEval benchmark and our dataset MultiEval.

Model	HumanEval pass@1	HumanEval EFR	MultiEval EFR
Qwen2.5-Coder-7B	84,8%	96,9%	69,4%
Our	**86,6%**	**98.2%**	**75,8%**

Furthermore, to achieve more representative results, both models were tested in an iterative pipeline, illustrated in Fig. 2, that involved improving the generated code based on feedback from the static code analyzer Svace.

Table 2 displays the results for both the fine-tuned and original models on the HumanEval dataset, including the pass@1 metric after two iterations of code patching, as well as the number of problems solved without errors in the first generaton, number of problems solved after the first iteration of code corrections using feedback from the static analyzer and the number of problems that were not resolved without errors after two iterations of code patching pipeline. On the second iteration, no improvements were observed for the original Qwen2.5-Coder-7B-instruct model, so it was not included in the table, although the iteration was actually conducted.

We tested the trained model on the Secure Coding Benchmark [4], on which we got an improvement in the vulnerable percentage metric, which is responsible for the percentage of test cases evaluated to be vulnerable across the language.

Table 3 contains the BLEU metric on MultiEval dataset that is used to determine how well generated code matches one reference code and vulnerable percentage metric for the original model.

Table 2. Information about the number of correctly generated codes and the pass@1 metric on the HumanEval benchmark, which consists of 164 tasks, after iterative code patching using Svace for both fine-tuned and original models.

Model	pass@1	First attempt	After patch	Didn't pass
Qwen2.5-Coder-7B	82,9%	159	4	1
Our	**87,2%**	**161**	**3**	**0**

Table 3. BLEU and Vulnerable Percentage metrics for original Qwen2.5-Coder-7B-Instruct and our model on MultiEval benchmark.

Language	Original model		Our	
	BLEU	Vulnerable %	BLEU	Vulnerable %
C	**10,9**	41,0	10,8	**38,3**
C++	10,6	23,9	**10,7**	**22,4**
C#	**13,9**	26,8	13,6	**26,0**
Java	17,1	53,3	**17,4**	53,3
JavaScript	**10,3**	39,4	10,2	**39,0**
PHP	**13,7**	**36,4**	13,4	42,6
Python	8,4	**28,2**	8,4	28,8
Rust	**14,7**	42,2	14,4	**41,7**

4.4 Evaluation of feedback mechanism

To evaluate the effectiveness of the developed system and its ability to improve the quality and security of the generated code, a series of experiments with various language models were conducted. The main evaluation metrics were pass@1 and EFR (Error-Free Rate). The following models participated in the experiments: CodeLlama-7b-hf, Mistral-7B-Instruct-v0.3, deepseek-coder-7b-instruct-v1.5, Mamba-Codestral-7B-v0.1, Nxcode-CQ-7B-orpo. The MultiEval dataset was used for the experiments.

Each model is tested twice: once before applying feedback from the analyzers, and the second time after 3 iterations of code correction [7]. The original work determined that three iterations were sufficient, as beyond this point, quality did not improve significantly but generation time increased. Feedback is generated using two tools: Svace (for detecting syntactic and logical errors) and Bandit (for finding security vulnerabilities). The experiments were conducted in three modes: Svace only, Bandit only, and a combination of both. When using Svace alone, the average share of error-free solutions (EFR) increased by 11.5%, indicating high feedback efficiency while improving code quality. However, the pass@1 functional metric showed a slight decrease of about 1%, especially for weak models such as CodeLlama and Mistral. This is due to the fact that when correcting errors, the logical integrity of the program is sometimes violated if changes are not made carefully enough. Stronger models such as deepseek-coder and Nxcode-CQ performed better. They have maintained or even slightly increased the pass@1 value, while significantly improving the EFR. This suggests that high-quality models are better at receiving detailed feedback and are able to maintain the logical structure of the code while improving it. Results of this experiment are shown in Table 4.

Table 4. Evaluation results before using Svace as a feedback tool and after.

Model	pass@1 before	pass@1 after	EFR before	EFR after
CodeLlama-7b-hf	29,3%	28,1%	91%	**97,6%**
deepseek-coder-7b-instruct-v1.5	72%	**73,2%**	97%	**100%**
Mistral-7B-Instruct-v0.3	34,8%	33,5%	78,1%	**100%**
Mamba-Codestral-7B-v0.1	34,2%	**37,8%**	75%	**98,8%**
Nxcode-CQ-7B-orpo	**78,1%**	**79,9%**	**97%**	**99,4%**

When using Bandit for security analysis, the results turned out to be less pronounced, since this tool focuses specifically on finding vulnerabilities, rather than on functional correctness. Nevertheless, Bandit proved to be useful in combination with Svace.

The average EFR value remained virtually unchanged, remaining at 99.4%, but there was a noticeable difference in the types of problems detected. Bandit has made it possible to identify and eliminate risks such as the use of unsafe functions, hard-coded secrets, and potential attack vectors through user input. Evaluation results across models are shown in Table 5.

The most significant effect was achieved with the simultaneous use of Svace and Bandit as shown in Table 6. This approach allows you to check the code for both functional correctness and vulnerabilities. The average EFR value increased by 12%, indicating a comprehensive improvement in code quality.

The pass@1 metric also showed a slight positive shift of about 1%, especially for models with a high initial accuracy level. This indicates that higher-quality models are able to effectively use multi-faceted feedback and maintain the logical integrity of the code while improving it.

The experimental results showed that all the tested models react differently to feedback from the analyzers. Stronger models such as deepseek-coder and Nxcode-CQ demonstrate good adaptability to code improvement and are able to maintain the logical integrity of the solution. Less powerful models such as Codestral and Mistral benefit less from the iterative process and may allow regressions when making changes. The integration of static analyzers into the code generation cycle

has significantly improved the quality and security of output solutions. The greatest effect is achieved with the combined use of Svace and Bandit, which provides comprehensive code verification.

Table 5. Evaluation results before using Bandit as a feedback tool and after.

Model	pass@1 before	pass@1 after	EFR before	EFR after
CodeLlama-7b-hf	29,3%	28,7%	91,4%	**96,4%**
deepseek-coder-7b-instruct-v1.5	72%	**71,3%**	97%	**99,4%**
Mistral-7B-Instruct-v0.3	34,8%	34,2%	78,1%	**96,7%**
Mamba-Codestral-7B-v0.1	34,2%	**34,2%**	75%	**94,4%**
Nxcode-CQ-7B-orpo	**78,1%**	**78,1%**	97%	**98,4%**

Table 6. Evaluation results before using Bandit and Svace as feedback tools and after.

Model	pass@1 before	pass@1 after	EFR before	EFR after
CodeLlama-7b-hf	29,3%	27,4%	91,4%	**97,6%**
deepseek-coder-7b-instruct-v1.5	72%	**72,6%**	97%	**100%**
Mistral-7B-Instruct-v0.3	34,8%	32,9%	78,1%	**100%**
Mamba-Codestral-7B-v0.1	34,2%	**37,8%**	75%	**98,8%**
Nxcode-CQ-7B-orpo	**78,1%**	**78,7%**	97%	**99,4%**

5. Conclusions

In this work, we tested an iterative pipeline with a fine-tuned model for improving the safety and reliability of generated code. Our experiments showed that, on average, only three iterations were required to eliminate most errors.

Furthermore, we enhanced the Qwen2.5-Coder-7B-instruct model through reinforcement learning using DPO. By fine-tuning the model on pairs of erroneous and correct code from the CodePreference dataset, we achieved a notable reduction in any errors.

These findings suggest that combining iterative feedback with advanced reinforcement learning techniques can significantly enhance the safety and reliability of LLM-generated code. Future work could explore the integration of additional static, dynamic, and security analysis tools, as well as the extension of this approach to other programming languages.

References

[1]. Mckenna G. Over 25pichai says it's just the start [Электронный ресурс] // Fortune. URL: https://fortune.com/2024/10/30/googles-code-ai-sundar-pichai/ (дата обращения: 01.05.2025).

[2]. Becker B. A. et al. Programming is hard-or at least it used to be: Educational opportunities and challenges of ai code generation //Proceedings of the 54th ACM Technical Symposium on Computer Science Education, vol. 1, 2023, pp. 500-506.

[3]. Li J. et al. Poison attack and defense on deep source code processing models //arXiv preprint, 2022. Available at: arXiv:2210.17029, accessed 09.10.2025.

[4]. Bhatt M. et al. Purple llama cyberseceval: A secure coding benchmark for language models //arXiv preprint, 2023. Available at: arXiv:2312.04724, accessed 09.10.2025.

[5]. Shaikhelislamov D., Drobyshevskiy M., Belevantsev A. LLM-based Interactive Code Generation: Empirical Evaluation //2024 Ivannikov Ispras Open Conference (ISPRAS). IEEE, 2024, pp. 1-5.

[6]. Siddiq M. L., Santos J. C. S. SecurityEval dataset: mining vulnerability examples to evaluate machine learning-based code generation techniques //Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security, 2022, pp. 29-33.

[7]. Shaikhelislamov D. S., Drobyshevskiy M. D., Belevancev A. A. Ensuring trustworthy code: leveraging a static analyzer to identify and mitigate defects in generated code //Записки научных семинаров ПОМИ, 2024, vol. 540, no. 0, pp. 233-251.

[8]. Liu J. et al. Learning code preference via synthetic evolution //arXiv preprint, 2024. Available at: arXiv:2410.03837, accessed 09.10.2025.

[9]. Pearce H. et al. Examining zero-shot vulnerability repair with large language models //2023 IEEE Symposium on Security and Privacy (SP). – IEEE, 2023. – С. 2339-2356.

[10]. Touvron H. et al. Llama 2: Open foundation and fine-tuned chat models //arXiv preprint, 2023. Available at: arXiv:2307.09288, accessed 09.10.2025.

[11]. Li H. et al. Enhancing static analysis for practical bug detection: An llm-integrated approach //Proceedings of the ACM on Programming Languages, 2024, vol. 8, no. OOPSLA1, pp. 474-499.

[12]. Kharma M. et al. Security and Quality in LLM-Generated Code: A Multi-Language, Multi-Model Analysis //arXiv preprint, 2025. Available at: arXiv:2502.01853, accessed 09.10.2025.

[13]. He J. et al. Instruction tuning for secure code generation //arXiv preprint, 2024. Available at: arXiv:2402.09497, accessed 09.10.2025.

[14]. Belevantsev A. et al. Design and development of Svace static analyzers //2018 Ivannikov Memorial Workshop (IVMEM), IEEE, 2018, pp. 3-9.

[15]. Tsiazhkorob U. V., Ignatyev V. N. Classification of Static Analyzer Warnings using Machine Learning Methods //2024 Ivannikov Memorial Workshop (IVMEM), IEEE, 2024, pp. 69-74.

[16]. He J., Vechev M. Large language models for code: Security hardening and adversarial testing //Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, 2023. pp. 1865-1879.

[17]. Liu M. et al. An empirical study of the code generation of safety-critical software using llms //Applied Sciences, 2024, vol. 14, no. 3, p. 1046.

[18]. Allal L. B. et al. A framework for the evaluation of code generation models [Online] // GitHub. Available at: https://github.com/bigcode-project/bigcode-evaluation-harness, accessed 09.10.2025.

[19]. Hui B. et al. Qwen2. 5-coder technical report //arXiv preprint, 2024. Available at: 4 arXiv:2409.12186, accessed 09.10.2025.

[20]. Zheng Q. et al. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x //Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2023, pp. 5673-5684.

[21]. [21]. Hendrycks D. et al. Measuring coding challenge competence with apps //arXiv preprint, 2021. Available at: arXiv:2105.09938, accessed 09.10.2025.

[22]. Babe H. M. L. et al. Studenteval: A benchmark of student-written prompts for large language models of code //arXiv preprint, 2023. Available at: arXiv:2306.04556, accessed 09.10.2025.

[23]. Du M. et al. Mercury: A code efficiency benchmark for code large language models //Advances in Neural Information Processing Systems, 2024, vol. 37, pp. 16601-16622.

[24]. Yin P. et al. Learning to mine aligned code and natural language pairs from stack overflow //Proceedings of the 15th international conference on mining software repositories, 2018, pp. 476-486.

[25]. Austin J. et al. Program synthesis with large language models //arXiv preprint, 2021. Available at: arXiv:2108.07732, accessed 09.10.2025.

[26]. Lai Y. et al. DS-1000: A natural and reliable benchmark for data science code generation //International Conference on Machine Learning. PMLR, 2023, pp. 18319-18345.
[27]. Liu J. et al. Learning code preference via synthetic evolution //arXiv preprint, 2024. Available at: arXiv:2410.03837, accessed 09.10.2025.
[28]. Bhandari G., Naseer A., Moonen L. CVEfixes: automated collection of vulnerabilities and their fixes from open-source software //Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering, 2021, pp. 30-39.
[29]. Rafailov, R., Sharma, A., Mitchell, E., Manning, CD., Ermon, S., Finn, C. Direct preference optimization: Your language model is secretly a reward model //Advances in neural information processing systems, 2023, vol. 36, pp. 53728-53741.

Информация об авторах / Information about authors

Данил Салаватович ШАЙХЕЛИСЛАМОВ – исследователь Института системного программирования, старший преподаватель Высшей школы экономики, аспирант Московского физико-технического института. Сфера научных интересов: большие языковые модели, генерация кода.

Danil Salavatovich SHAIKHELISLAMOV – researcher at the Institute of System Programming, senior lecturer at the Higher School of Economics, postgraduate student at the Moscow Institute of Physics and Technology. His research interests include large language models, code generation.

Мария Сергеевна ВАРЕЦА – студентка МИРЭА. Сфера научных интересов: большие языковые модели, генерация кода.

Maria Sergeevna VARETSA – student MIREA. His research interests include security technologies and business informatics.

Арсений Сергеевич СЁМКИН – студен ВШЭ. Сфера научных интересов: большие языковые модели, программирование.

Arseny Sergeevich SYOMKIN – student at HSE University. His research interests include large language models and software engineering.

Олег Юрьевич РОГОВ – старший научный сотрудник, руководитель группы «Доверенные и безопасные интеллектуальные системы», Институт искусственного интеллекта; научный сотрудник лаборатории вычислительного интеллекта, Сколковский институт науки и технологий (СколТех).

Oleg Yurievich ROGOV – Senior Researcher, Head of the Trusted and Secure Intelligent Systems Group, AIRI Institute of Artificial Intelligence; Researcher at the Computational Intelligence Laboratory, Skolkovo Institute of Science and Technology (Skoltech).

DOI: 10.15514/ISPRAS-2025-37(5)-9

Интерактивная генерация кода на основе LLM: эмпирическая оценка

[1,2] *Д.С. Шайхелисламов, ORCID: 0000-0002-9734-7937 <shaykhelislamov.ds@ispras.ru>*
[1,2] *М.Д. Дробышевский, ORCID: 0000-0002-1639-9154 <drobyshevsky@ispras.ru>*
[1,3] *А.А. Белеванцев, ORCID: 0000-0003-2817-0397 <abel@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.*

[2] *Московский физико-технический институт, Россия, 141700 Московская область, г. Долгопрудный, Институтский переулок, 9.*

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

Аннотация. ИИ-помощники разработчика, основанные на больших языковых моделях (LLM), продемонстрировали большие возможности в генерации программ по текстовому описанию. Однако в таком коде зачастую встречаются ошибки. Пользователи ожидают код без дефектов и, в идеале, четкие указания на их присутствие. Проверенный код может снизить потенциальные бизнес-риски, связанные с внедрением сгенерированного кода. Используя расширение CodePatchLLM, в работе оценивается качество генерируемых программных решений. Эксперименты показывают, что даже одна итерация исправления кода для языка Java во всех наборах данных и моделях снижает на 19,1% количество дефектов при сохранении функциональной корректности.

Ключевые слова: большая языковая модель; проверка кода; безопасный код.

Для цитирования: Шайхелисламов Д.С., Дробышевский М.Д., Белеванцев А.А. Интерактивная генерация кода на основе LLM: эмпирическая оценка. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 123–130. DOI: 10.15514/ISPRAS–2025–37(5)–9.

LLM-based Interactive Code Generation: Empirical Evaluation

[1,2] *D.S. Shaikhelislamov, ORCID: 0000-0002-9734-7937 <shaykhelislamov.ds@ispras.ru>*
[1,2] *M.D. Drobyshevskiy, ORCID: 0000-0002-1639-9154 <drobyshevsky@ispras.ru>*
[1,3] *A.A. Belevantsev, ORCID: 0000-0003-2817-0397 <abel@ispras.ru>*

[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, Institutsky lane 9, Dolgoprudny, Moscow region, 141700, Russia.*

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

Abstract. Recently, large language models (LLMs), those pretrained on code, have demonstrated strong capabilities in generating programs from informal natural language intent. However, LLM-generated code is prone to bugs. Developers interacting with LLMs seek trusted code and, ideally, clear indications of potential bugs and vulnerabilities. Verified code can mitigate potential business risks associated with adopting generated code. We use model-agnostic framework CodePatchLLM, an extension for LLM that utilizes Svace feedback to enhance code generation quality. We evaluate CodePatchLLM on four popular LLMs across three datasets. Our experiments show an average absolute reduction of 19.1% in static analyzer warnings for Java across all datasets and models, while preserving pass@1 code generation accuracy.

Keywords: large language model; code verification; trusted code.

For citation: Shaikhelislamov D.S., Drobyshevskiy M.D., Belevantsev A.A. LLM-based Interactive Code Generation: Empirical Evaluation. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 123-130 (in Russian). DOI: 10.15514/ISPRAS-2025-37(5)-9.

1. Введение

ИИ-помощники разработчика на основе LLM быстро превратились из инструментов предсказания следующего символа в расширения для написания фрагментов программы. Опрос разработчиков, проведенный платформой StackOverflow, показал, что 70% респондентов используют или планируют использовать инструменты для кодирования с использованием искусственного интеллекта в этом году [1]. ServiceNow объявила, что их решение «преобразование текста в код», основанное на тонкой настройке StarCoder [2], приводит к увеличению производительности разработчиков на 52%. В работе рассматривается применение ИИ-помощника, когда система на текстовое описание возвращает код (рис. 1).

Сгенерированный код подвержен ошибкам, которые можно легко не заметить [3]. Рассмотрим простой запрос к GPT-3.5: «*Напиши функцию на Pyhton, которая проверяет, что скрипт существует*». Модель может вернуть код со строкой «`result=subprocess.run(['bash', script])`». Однако известно, что этот код подвержен дефекту CWE-78. Исходя из этого, рекомендуется проверять сгенерированный код с помощью инструментов автоматического обнаружения дефектов [4]. Результаты проверки является сигналом пользователю о безопасности решения. Обратной связи только от модульных тестов и компилятора недостаточно для надежности программы. Инструменты статического анализа проводят более тщательную проверку исходного кода, чем компиляторы, которые обычно обнаруживают только синтаксические ошибки. Тесты на тысячи задач LeetCode для типизированных языков программирования выявили дефекты в 12% решениях [4]. Поскольку LLM могут допускать ошибки, пользователям было бы полезно получить подтверждение того, что сгенерированный код на самом деле правильный. А если нет, то получить исправленную версию. Вместе с этим, показатели, если они хорошо согласуются с фактической корректностью, должны быть независимы от выбранной модели.

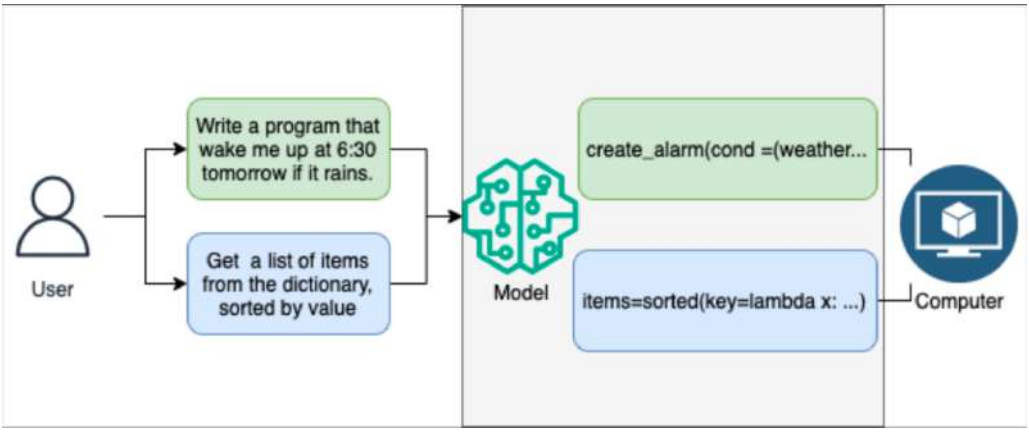


Рис. 1. Сгенерированный код по текстовому описанию зачастую используется без проверок, что может привести к включению в проект или запуску вредоносного кода в системе.

Fig. 1. The generated code may be potentially dangerous to use in an environment if it has not been previously checked for weaknesses and bugs.

С этой целью в данной работе стремимся понять: уменьшает ли фреймворк CodePatchLLM [5] дефекты в сгенерированном коде? Применим ли сообщения в качестве независимого сигнала о наличии дефектов в сгенерированном решении? CodePatchLLM, используя различные критерии корректности, проверяет решение на дефекты с помощью Svace [6] и применим для различных моделей. Насколько нам известно, ранее не предпринималось попыток использовать обратную связь от статического анализатора для исправления кода без изменения самой модели. В работе было сделано несколько выводов:

- Код сгенерированный LLMs и оцененный с использованием стандартных представлений о корректности кода, плохо работает на реальных наборах данных при использовании в задачах завершении и синтеза кода. Мы наблюдаем дефекты во всех рассмотренных языках программирования (Java, Python);
- Эксперименты с CodePatchLLM на различных задачах по программированию (MBPP, HumanEval, Leetcode) и моделях (CodelLama, CodeGen2, CodeX, GPT-3.5) показали, что инструмент может быть использован разработчиками, в качестве дополнительного сигнала о качестве сгенерированного решения.

В работе рассматривается проблема использования ИИ-помощника разработчика для генерации безопасного кода или информировании пользователя о том, что в сгенерированном решении присутствуют дефекты.

2. Сопутствующие работы

В этом разделе дан краткий обзор литературы по оценке генерации кода и использованию внешних инструментов для оценки качества.

2.1 Оценка генерации кода

Обычно для тестирования кода используются модульные тесты, собранные из GitHub и StackOverflow [7]. Такие тесты сконцентрированы на оценку функциональной корректности программы. Авторы в работе [8] вводят оценку на основе обратной связи компилятора, которая позволяет оценивать выполнимость генерируемого решения. В данной работе фокус на изучении того, могут ли LLM сопоставлять инструкции, полученные от статического анализатора, с кодом и использовать эти инструкции для корректировки программ на Java, Python.

2.2 Инструмент статического анализа

Другие инструменты статического анализа, такие как CodeQL [9], Semgrep [10], FlawFinder [11], также используются для обнаружения дефектов. Но эти инструменты не так многофункциональны и эффективны, как Svace. Инструмент позволяет одинаково взаимодействовать с программами на разных языках программирования и соответствует ГОСТ Р 71207-2024. Потенциально любые другие средства проверки программы могут быть реализованы в CodePatchLLM и могут использоваться совместно с другими.

2.3 Инструктирование для генерации кода

Методы инструктирования широко применяются в задачах, связанных с генерацией кода. Несколько исследований [12] посвящены использованию инструкций для улучшения качества генерируемого кода. В [13] подсказки используются для облегчения объяснения кода и разработки тестов. В нашем подходе особое внимание уделяется использованию инструментов анализа программ для повышения компилируемости и безопасности сгенерированного кода при сохранении эффективности решения проблем. В частности, мы используем обратную связь от компилятора и статического анализатора в качестве сигналов для модели.

3. Исследовательские вопросы

ИИ-помощники разработчика широко используются для синтеза функций и задач генерации кода. Формально, моделируется программа $\boldsymbol{x}$ как последовательность токенов. На шаге t модель вычисляет вероятность токена, учитывая его предыдущие токены $P(x_t\,|x_1, \dots, x_{t-1})$, где $x_1, \dots, x_{t-1}$ обеспечивают контекст. Энтропия $H(x_t|x_1, \dots, x_{t-1})$ измеряет неопределенность в отношении x_t с учетом контекста. Более длинная инструкция предоставляет больше токенов для проверки, потенциально снижая энтропию, если дополнительные токены содержат соответствующую информацию: $H(x_t|x_1, \dots, x_{t-1}) < H(x_t|x_1, \dots, x_{t-k})$, где $k < t - 1$. Это неравенство следует из неравенства обработки данных в теории информации, которое гласит, что использование дополнительных переменных (токенов) не может увеличить неопределенность.

Исследовательский вопрос 1. Насколько сгенерированный код обеспечивает защиту от дефекта в обычных задачах генерации, таких как синтез функций и восстановление программ?

Цель состоит в том, чтобы оценить, можно ли доверять коду, созданному LLM, с точки зрения безопасности, или же он содержит общие недостатки, которые могут нарушить целостность программного обеспечения.

Исследовательский вопрос 2. Можно ли повысить безопасность сгенерированного кода с помощью инструментов проверки?

Цель состоит в том, чтобы определить, могут ли рефлексивные подсказки, основанные на обратной связи от компилятора и статического анализатора, улучшить корректность кода и безопасность. Для каждого сгенерированного решения агрегируется обратная связь от компилятора или статического анализа и подается обратно в модель. Оценивается насколько исправления соответствуют корректности и безопасности, что оценивается через уменьшение числа обнаруженных дефектов.

Исследовательский вопрос 3. Можно ли повысить безопасность сгенерированного кода без ущерба для его качества?

В этом вопросе оценивается механизм обратной связи для повышения безопасности без ущерба функциональной корректности.

4. Эксперименты

Рассматриваются две генеративные задачи: синтез функций и дополнение функций по текстовому описанию. Эти задачи являются популярными приложениями ИИ- помощников разработчика, которые применяются, например, в Github Copilot.

4.1 Задачи и наборы данных

Для проверки используется базовая настройка CodePatchLLM, включая описания подсказок и количество итераций обновления, что описано в работе [5].

Синтез функции: задача направлена на создание функции по описанию. В работе используются популярные для этих целей наборы данных HumanEval-X [14]. Набор состоит из 164 примеров на Java и Python.

Дополнение функции: по текстовому описанию и сигнатуре необходимо реализовать функцию. Для данной задачи используются наборы Leetcode [5] и MBPP [15] (MBJP для Java). Эти данные собраны в табл. 1.

Табл. 1. Рассматриваемые задачи и наборы данных.
Table 1. List of tasks and associated datasets.

Задача	Набор данных	Количество задач
Синтез функции	HumanEval-X	164
Дополнение функции	MBPP (MBJP для Java)	880
	Leetcode	2 612

4.2 Метрика

Функциональная корректность. Для оценки сгенерированных кодов используется метрика *pass@1*, которая вычисляется, как процент задач, по которым были пройдены все тесты, используя 1 сгенерированный код для каждой задачи.

Безопасность кода. Будет определять код для задачи безопасным, если после проверки статическим анализатором, сгенерированное решение не содержит ошибки и другие диагностические предупреждения от компилятора. Таким образом задача сводится к повышению безопасности без снижения функциональной корректности.

4.3 Модели

Для экспериментов используются четыре популярные публичные модели. К ним относятся CodelLama [16], OpenAI GPT-3.5, OpenAI CodeX [17] и CodeGen2-16B [18].

5. Результаты

В исследовании CodePatchLLM используются в задачах для генерации программ на Python и Java. В табл. 2 приведена оценка функциональной корректности моделей по различным задачам. В среднем, CodePatchLLM не влияет на показатель pass@1 для сгенерированного кода. Однако для некоторых моделей это может привести к увеличению проходного балла на 1 балл.

Исследовательский вопрос 1.

Табл. 3 показывает, что CodeGen2 и CodeLlama приводят к относительно низкому проценту дефектов в коде в Python, составляющему 1% и 0,4% соответственно, по сравнению с CodeX (1,5%) и GPT-3,5 (2,1%). CodePatchLLM эффективно устраняет 90% обнаруженных уязвимостей, используя обратную связь в виде инструкций.

Табл. 2. Оценка функциональной корректности.
Table 2. Performance comparison of models.

	Pass@1					
	Python			Java		
	HumanEval	MBPP	Java	HumanEval	MBPP	Java
CodeGen2	23,1	29,1	7,8	20,3	21,9	6,0
+CodePatchLLM					22,0	3,9
CodeX	47,2	61,8	10,5	20,4	38,1	9,2
+CodePatchLLM					37,0	10,0
GPT3.5	64,6	72,0	29,0	62,4	44,8	28,0
+CodePatchLLM				50,3	45,2	29,6
CodeLlama	81,7	65,6	36,0	78,5	40,0	12,0
+CodePatchLLM				75,4	43,2	25,0

Табл. 3. Доля задач с дефектами в решениях для разных моделей и языков программирования.
Table 3. The percentage of compilation errors and weakness detected and fixed by CodePatchLLM..

	Ошибки компиляции и дефекты			
	CodeGen2	CodeX	GPT3.5	CodeLlama
Python	1,0%	1,5%	2,1%	0,4%
Java	10%	37,9%	16,2%	12,5%

Для Java процент дефектов заметно выше во всех моделях, при этом CodeX генерирует наибольшее количество дефектов – 37,9%. Относительно низкий уровень дефектов в коде Python позволяет предположить, что эти модели в целом генерируют более безопасный код для Python, чем Java. Это различие может быть связано со структурой языка Python или, возможно, со смещённостью данных для обучения и тонкой настройкой в сторону программ на Python [19]. Число дефектов в Java значительно выше, особенно для CodeX и GPT-3.5, причем CodeX показывает почти 38%. Это несоответствие может указывать на то, что связанные с Java наборы данных или методы обучения для этих моделей нуждаются в доработке, или что более строгая типизация и структура Java выявляют больше недостатков при анализе Svace.

Исследовательский вопрос 2.

CodePatchLLM демонстрирует среднее абсолютное снижение предупреждений статического анализатора по всем наборам данных и моделям на 19,1%. Учитывая более высокий базовый уровень дефектов в Java, применение корректирующей обратной связи CodePatchLLM может быть особенно полезным для кода, сгенерированного на Java.

Исследовательский вопрос 3.

Для задач на Python CodePatchLLM не оказывает существенного влияния на функциональную корректность сгенерированного кода, измеряемого по показателю pass@1. Однако, несмотря на это, наблюдаются незначительные дефекты и предупреждения, о чем сообщает статический анализатор.

В среднем, использование CodePatchLLM не оказывает существенного влияния на функциональную корректность по метрике pass@1 во всех моделях (таблица 2). Однако отдельные модели показывают разные результаты, при этом некоторые из них выигрывают от изменений, а другие остаются в основном неизменными. Этот вывод свидетельствует о том, что, хотя CodePatchLLM не может улучшить функциональную корректность, он может повысить безопасность сгенерированного решения, снижая количество дефектов в коде.

6. Выводы

В этом исследовании исследовался вопрос безопасность сгенерированного решения ИИ-помощником разработчика для решения распространенных задач программирования. Исследовательские вопросы в работе касались повышения безопасности кода без ущерба для функциональной корректности. Результаты показывают, что сгенерированные решения ИИ-помощниками разработчика, могут содержать дефекты, особенно при решении сложных задач. Это подчеркивает важность интеграции дополнительных процессов проверки обнаружения дефектов и исправления в процессе генерации кода. Более того, эксперименты показывают, что возможно повысить безопасность сгенерированного кода без снижения его функционального качества. В случаях, где повышение безопасности автоматизированным способом невозможно, сигналы о наличии дефектов в коде, позволяют пользователю быть информированным об их присутствии, что, в свою очередь, увеличивает общее доверие к использованию ИИ-помощника разработчика.

В частности, использование CodePatchLLM позволило снизить количество предупреждений статического анализатора в среднем на 19,1% без негативного влияния на функциональную корректность кода. Для некоторых моделей использование CodePatchLLM даже привело к небольшому увеличению их корректности.

Ограничения и будущие работы

CodePatchLLM эффективен для ИИ-помощников разработчика на основе LLM, которые широко используются на практике. Однако фреймворк не корректирует саму модель. Модели могут допускать одни и те же ошибки при использовании. Кроме того, CodePatchLLM не рассматривает случаи, когда модель должна использовать существующие в проекте классы и методы [20]. Рассмотрение сценариев переобучение модели на собранном наборе ошибок и исправлений представляет собой многообещающее направление будущей работы. Наконец, отмечается, что CodePatchLLM не предоставляет гарантий повышения безопасности сгенерированного кода, так как это во многом зависит от используемых инструментов проверки.

Список литературы

[1]. StackOverflow, Developer Survey. Доступно по ссылке: https://survey.stackoverflow.co/2023/#ai-tools-in-the-development-process, обращение 30.05.2023.

[2]. Li R., Allal L.B., Zi Y., Muennighoff N., Kocetkov D., Mou C., Marone M., Akiki C., Li J., Chim J. Starcoder: may the source be with you! //arXiv preprint, 2023. Доступно по ссылке: arXiv:2305.06161, обращение 10.10.2025.

[3]. Tambon F., Moradi-Dakhel A., Nikanjam A., Khomh F., Desmarais MC., Antoniol G. Bugs in large language models generated code: An empirical study // Empirical Software Engineering, 2025, vol. 30, no. 3, p. 65.

[4]. Shaikhelislamov D., Drobyshevskiy M., Belevantsev A. LLM-based Interactive Code Generation: Empirical Evaluation // 2024 Ivannikov Ispras Open Conference (ISPRAS). IEEE, 2024, pp. 1-5.

[5]. Shaikhelislamov D. S., Drobyshevskiy M. D., Belevancev A. A. Ensuring trustworthy code: leveraging a static analyzer to identify and mitigate defects in generated code // Записки научных семинаров ПОМИ, 2024, vol. 540, no. 0, pp. 233-251.

[6]. Belevantsev A., Borodin A., Dudina I., Ignatiev V., Izbyshev A., Polyakov S. Design and development of Svace static analyzers // 2018 Ivannikov Memorial Workshop (IVMEM). IEEE, 2018, pp. 3-9.

[7]. Agashe R., Iyer S., Zettlemoyer L. JuICe: A large scale distantly supervised dataset for open domain context-based code generation // arXiv preprint, 2019. Доступно по ссылке: arXiv:1910.02216, обращение 10.10.2025.

[8]. Grubisic D., Cummins C., Seeker V., Leather H. Compiler generated feedback for large language models //arXiv preprint, 2024. Доступно по ссылке: arXiv:2403.14714, обращение 10.10.2025.

[9]. Avgustinov P., Moor O., Jones MP., Schäfer M. QL: Object-oriented queries on relational data // 30th European Conference on Object-Oriented Programming (ECOOP 2016). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016, pp. 2: 1-2: 25.

[10]. Semgrep, 2023. [Online]. Available at: https://semgrep.dev/, обращение 10.10.2025.
[11]. FlawFinder, 2023. [Online]. Available at: https://dwheeler.com/flawfinder, обращение 10.10.2025.
[12]. Li H., Hao Y., Zhai Y., Qian Z. Enhancing static analysis for practical bug detection: An llm-integrated approach // Proceedings of the ACM on Programming Languages. 2024, vol. 8, no. OOPSLA1, pp. 474-499.
[13]. Zhang T, Yu T., Hashimoto T., Lewis M., Yih W., Fried D., Wang S. Coder reviewer reranking for code generation //International Conference on Machine Learning. PMLR, 2023, pp. 41832-41846.
[14]. Zheng Q., Xia X., Zou X., Dong Y., Wang S., Xue Y., Shen L., Wang Z., Wang A., Li Y., Su T., Yang Z., Tang J. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x // Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2023, pp. 5673-5684.
[15]. Odena, A., Sutton, C., Dohan, D. M., Jiang, E., Michalewski, H., Austin, J., Bosma MP., Nye M. Program synthesis with large language models //arXiv preprint, 2021. Доступно по ссылке: arXiv:2108.07732, обращение 10.10.2025.
[16]. Rozière B., Gehring J., Gloeckle F., Sootla S., Gat I., Ellen Tan X., Adi Y., Liu J., Sauvestre R., Remez T., Rapin J., Kozhevnikov A., Evtimov I., Bitton J., Bhatt M., Ferrer CC., Grattafiori A., Xiong W., Défossez A., Copet J., Azhar F., Touvron H., Martin L., Usunier N., Scialom T., Synnaeve G. Code llama: Open foundation models for code //arXiv preprint, 2023. Доступно по ссылке: arXiv:2308.12950, обращение 10.10.2025.
[17]. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan [и др.]. Evaluating large language models trained on code //arXiv preprint, 2021. Доступно по ссылке: arXiv:2107.03374, обращение 10.10.2025.
[18]. Nijkamp, E., Hayashi, H., Xiong, C., Savarese, S., & Zhou, Y. Codegen2: Lessons for training llms on programming and natural languages //arXiv preprint, 2023. Доступно по ссылке: arXiv:2305.02309, обращение 10.10.2025.
[19]. Siddiq, M. L., Dristi, S., Saha, J., & Santos, J. C. The fault in our stars: Quality assessment of code generation benchmarks // 2024 IEEE International Conference on Source Code Analysis and Manipulation (SCAM). IEEE, 2024, pp. 201-212.
[20]. Liao, D., Pan, S., Sun, X., Ren, X., Huang, Q., Xing, Z. [и др.]. A 3-codgen: A repository-level code generation framework for code reuse with local-aware, global-aware, and third-party-library-aware // IEEE Transactions on Software Engineering. 2024.

Информация об авторах / Information about authors

Данил Салаватович ШАЙХЕЛИСЛАМОВ – исследователь Института системного программирования, старший преподаватель Высшей школы экономики, аспирант Московского физико-технического института. Сфера научных интересов: большие языковые модели, генерация кода.

Danil Salavatovich SHAIKHELISLAMOV – researcher at the Institute of System Programming, senior lecturer at the Higher School of Economics, postgraduate student at the Moscow Institute of Physics and Technology. His research interests include large language models, code generation.

Михаил Дмитриевич ДРОБЫШЕВСКИЙ – кандидат физико-математических наук, научный сотрудник ИСП РАН. Сфера научных интересов: доверенный ИИ, объяснимый ИИ.

Mikhail Dmitrievich DROBYSHEVSKIY – Cand. Sci (Phys.-Math), researcher at ISP RAS. Research interests: trusted AI, explainable AI.

Андрей Андреевич БЕЛЕВАНЦЕВ – доктор физико-математических наук, член-корреспондент РАН, ведущий научный сотрудник ИСП РАН, профессор кафедры системного программирования ВМК МГУ. Сфера научных интересов: статический анализ программ, оптимизация программ, параллельное программирование.

Andrey Andreevich BELEVANTSEV – Dr. Sci. (Phys.-Math.), Prof., corresponding Member RAS, leading researcher at ISP RAS, Professor at Moscow State University. Research interests: static analysis, program optimization, parallel programming.

DOI: 10.15514/ISPRAS-2025-37(5)-10

Application of SVAN Static Analysis Tool on Open RTL Benchmarks

S.M. Panova, ORCID: 0009-0008-5106-0915 <panova@ispras.ru>
S.A. Smolov, ORCID: 0000-0003-0173-3081 <smolov@ispras.ru>
M.M. Volkova, ORCID: 0009-0009-8324-7562 <volchonok03@bk.ru>

Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
Plekhanov Russian University of Economics,
36, Stremyanny lane, Moscow, 115004, Russia.

Abstract. The article presents an experimental evaluation of SVAN, a static analysis tool designed for functional verification of RTL models written in Verilog and SystemVerilog. The research addresses the growing need for reliable domestic EDA tools, particularly in light of restrictions on proprietary solutions. SVAN's architecture integrates formal methods and heuristic approaches to detect a wide range of errors, including syntactic issues, coding style violations, logical inconsistencies, and security vulnerabilities. Empirical testing on open-source hardware benchmarks demonstrates SVAN's superior effectiveness compared to Synopsys VCS and Verilator, with a 73% broader error detection spectrum and 25-23% higher error identification rate, respectively. Key advantages of SVAN include high analysis accuracy and detailed error classification. However, limitations such as reduced flexibility in handling mixed-language designs highlight areas for future improvement. The study underscores SVAN's potential as a competitive tool for static verification in electronic design automation.

Keywords: static analysis; RTL; SVAN; Verilog; SystemVerilog; functional verification; error detection; open-source benchmarks.

For citation: Panova S.M., Smolov S.A., Volkova M.M. Application of SVAN Static Analysis Tool on Open RTL Benchmarks. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 131-142. DOI: 10.15514/ISPRAS-2025-37(5)-10.

Acknowledgements: The project is supported by the Ministry of Industry and Trade of the Russian Federation within the R&D project "Development of a static analysis system for a hardware description language", code "CAD-Analysis" (as part of the R&D project "CAD microelectronics", the main contractor is JSC "ISTC MIET").

Применение инструмента SVAN статического анализа описаний аппаратуры для верификации открытых тестовых наборов

С.М. Панова, ORCID: 0009-0008-5106-0915 <panova@ispras.ru>
С.А. Смолов, ORCID: 0000-0003-0173-3081 <smolov@ispras.ru>
М.М. Волкова, ORCID: 0009-0009-8324-7562 < volchonok03@bk.ru>

Институт системного программирования им. В.П. Иванникова РАН,
Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.
Российский экономический университет им. Г.В. Плеханова,
Россия, 115054, г. Москва, Стремянный переулок, д.36.

Аннотация. В статье представлены результаты экспериментального анализа инструмента SVAN статического анализа описаний цифровой аппаратуры на языках Verilog и SystemVerilog. Инструмент разрабатывается в ИСП РАН и предоставляет средства формального и эвристического анализа HDL-описаний, нацеленные на выявление синтаксических ошибок, нарушений стиля оформления кода, проблем безопасности. Эксперименты, проведенные на описаниях из открытого тестового набора hdl-benchmarks, демонстрируют более высокую эффективность SVAN в сравнении с открытым инструментом Verilator и проприетарным инструментом Synopsys VCS. В частности, SVAN обнаружил на 73% больше типов ошибок и на 23-25% больше ошибок в целом. Ключевые преимущества инструмента SVAN состоят в более высоком уровне локализации ошибок и развитой типологии ошибок. К выявленным недостаткам инструмента SVAN относится ограниченная поддержка RTL-моделей, в которых используется несколько языков описания. Полученные результаты подчеркивают потенциал SVAN как конкурентоспособного инструмента статического анализа в области автоматизации проектирования цифровой аппаратуры.

Ключевые слова: статический анализ; RTL-модель; HDL-описание; анализатор SVAN; языки описания аппаратуры Verilog, SystemVerilog; функциональная верификация; обнаружение ошибок; открытый тестовый набор.

Для цитирования: Панова С.М., Смолов С.А., Волкова М.М. Применение инструмента SVAN статического анализа описаний аппаратуры для верификации открытых тестовых наборов. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 131–142 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)-10.

Благодарности: Проект финансируется Министерством промышленности и торговли Российской Федерации в рамках ОКР «Разработка системы статического анализа для языка описания аппаратуры», шифр «САПР-Анализ» (в составе ОКР «САПР микроэлектроника», головной исполнитель – АО «МНТЦ МИЭТ»).

1. Introduction

The design of ultra-large-scale integrated circuits (ULSICs) is a complex task that requires high precision and reliability at all stages of development. In the early stages of designing digital ULSICs, they are described at the Register Transfer Level (RTL) using Hardware Description Languages (HDLs), such as Verilog [1] and SystemVerilog [2]. Modern RTL models can reach significant sizes, encompassing millions of lines of code [3], making automated functional verification as one of the key tasks in the development process. Functional verification is aimed at detecting design errors that could lead to incorrect operation of the ULSIC or its complete failure.

To solve the problem of functional verification, various methods are employed, each possessing their own advantages and limitations. Static analysis holds a special place among them, as a technique for examining HDL description without its simulation. This approach includes the analysis of the structure, syntax, and semantics of the source code, making applicable to identifying a wide range of issues, from syntactic errors to potential vulnerabilities [4].

Static analysis tools implement the following processing stages:

- *Code Analysis*. The source code is transformed into an abstract syntax tree (AST) or another internal representation (IR), enabling the tool to comprehend the structure of the code.
- *Rule Application*. A set of predefined rules or checks is applied to the IR, aimed at identifying issues of a specific type, such as syntactic errors, style violations, or logical inconsistencies.
- *Problem Reporting*. Upon detecting rule violations, the tool generates a report containing the exact location of the issue (usually, a file name, and a line number), a description of the problem, and, probably, a recommendation for its resolution.

Currently, there are both commercial and open-source tools available for static analysis of RTL models. Commercial tools include VCS [5] and SpyGlass [6] (Synopsys), JasperGold (Cadence) [7], and Questa (Siemens) [8]. Among the open-source tools for static analysis are Slang [9], Verilator [10], Svlint [11], Surelog [12], and Verible [13]. Open-source tools often exhibit limited functionality compared to their commercial counterparts. For example, they may not support the full range of HDL standards or demonstrate lower accuracy in detecting complex errors. Proprietary static analysis tools are currently either unavailable or restricted for use under sanction regimes, posing significant risks to the domestic electronics industry. This situation forces the industry to either adapt existing tools or completely abandon their use. Furthermore, the utilization of foreign EDA (Electronic Design Automation) tools carries inherent information security risks.

There are studies in the literature demonstrating the application of these tools [14-15]; however, experimental comparison of their effectiveness has not been conducted previously, making this task novel and relevant for scientific investigation.

Taking into account the above, the development of domestic EDA tools represents a strategic task that will ensure technological independence, information security, and the sustainable growth of the country's electronics industry. Currently, the Ivannikov Institute for System Programming of the Russian Academy of Sciences (ISP RAS) is conducting research and development work aimed at creating SVAN, a static analysis tool for RTL models written in Verilog and SystemVerilog. SVAN tool development is supported by the Ministry of Industry and Trade of the Russian Federation.

The aim of this research is to apply the SVAN tool, currently under development [16], for the functional verification of open-source RTL models. The target design models selected for this purpose were benchmarks – sets of HDL descriptions originally intended for testing digital VLSI EDA tools, as well as for conducting comparative studies in this domain [17-18].

To achieve this goal, the following tasks were identified. First, an analysis of the applicability of SVAN for collection [19] of open-source RTL benchmarks was made. Next, the same benchmarks were analyzed using proprietary Synopsys VCS tool and open-source Verilator tool. After that, the results comparison was made.

A comparative analysis of open-source tools constitutes an independent scientific problem that falls outside the scope of the current study and has not been previously addressed in the literature. For the purposes of this comparison, we selected Verilator project – a widely recognized open-source tool that has been under active development since 2019 (currently in its sixth year of development). The tool’s popularity and broad acceptance within the community served as the primary criteria for its selection.

As the second tool in our comparison, we chose Synopsys VCS – a well-established commercial solution that is highly regarded in the industry and was accessible to us during the study. Both tools were selected due to their strong market presence, long-term development history, and widespread adoption in industrial applications.

An extended comparison of open-source tools may be pursued as a subject of future research and more comprehensive analysis.

2. SVAN tool description

Static analysis tools are designed to identify various issues in the code. These issues can be categorized into several classes.

1) *Simple syntactic* errors arise due to incorrect code writing, including the following:
   1). missing commas, semicolons, parentheses, etc.;
   2). incorrect use of language keywords or constructs;
   3). typo errors in variable names or function calls;
2) *Style violations* are issues related to coding standards [20-21], for example:
   1). inconsistent indentations;
   2). deprecated constructions usages;
   3). violations of naming conventions.
3) *Logical inconsistencies* represent a more complex class of issues that affect the behavior of the program or the design of the hardware. Examples of such inconsistencies include unreachable code (a situation where a portion of the code can never be executed due to logical errors) or race conditions (where the order of execution of code blocks directly impacts the overall outcome of the program).

Static analysis can also identify security-related issues, such as the use of uninitialized variables, which may lead to unpredictable behavior or improper handling of sensitive data, such as passwords or cryptographic keys.

The static analysis tool SVAN (Static Verification ANalysis tool) is a modern solution for verifying RTL models. SVAN supports Verilog (IEEE 1364-2005 standard) and SystemVerilog (IEEE 1800-2023 standard) HDL, enabling its use for verifying projects of varying complexity and purpose. The tool is designed to detect errors across various categories, which include the classes described above.

The architecture of SVAN is built on the principles of modularity and extensibility and is described on Figure 1. The tool includes the following key components: 1) SystemVerilog compiler (includes lexical analyzer, syntax analyzer and source code handling module); 2) analysis module (include rule detection module).

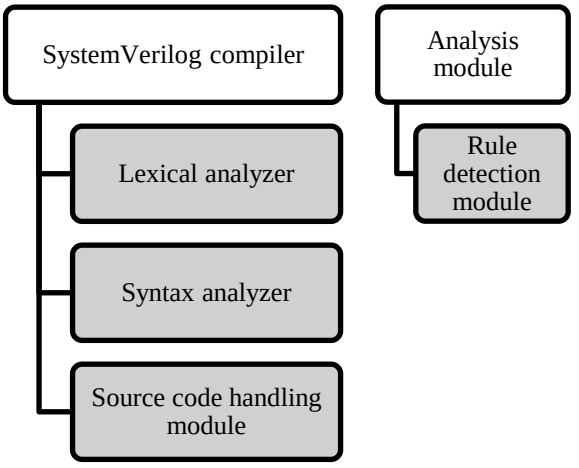


Fig. 1. SVAN Architecture.

The compiler is the central element of the SVAN architecture. It is responsible for processing the input code written in SystemVerilog and Verilog. The base compiler used is Slang – a modern, high-performance open-source compiler that supports the latest versions of the SystemVerilog standards (e.g., IEEE 1800-2023).

The advantages of using Slang lie in its ability to accurately analyze complex and large-scale projects. After processing the input code, the compiler transforms it into a specific IR, which serves as the input data for subsequent analysis stages.

The analysis module is the core of SVAN's functionality. It contains rule-based detectors designed to identify specific types of errors in the code. Each detector is focused on a particular category of issues, such as logical errors, coding style violations, or potential vulnerabilities.

All rules utilized by the tool have been classified into several categories based on language-related aspects and the development process. This classification ensures a systematic approach to identifying and addressing various types of issues in the code. Among the examples of categories are:

1) Assign – checks for assignment operations
2) Case – checks for case statements
3) Loop – checks for loop operators
4) Range – checks for out-of-bounds violations
5) Type – checks for type-related conflicts, and others

The detectors in the tool are implemented using two primary approaches: formal methods and heuristics.

To evaluate the performance and reliability of SVAN, it was tested on several well-known open-source hardware projects, including PicoRV32 [22], CVA6 [23], and OpenTitan [24]. The testing process involved analyzing these projects to identify potential issues across various categories, such as logical inconsistencies, coding style violations, and structural errors. The results of the testing confirm that SVAN is a reliable and efficient static analysis tool. Its ability to effectively handle complex real-world projects demonstrates its suitability for industrial use. By successfully processing large and intricate codebases, SVAN has proven its capability to meet the demands of modern hardware design verification, making it a valuable asset for both academic research and commercial applications in the field of electronic design automation (EDA).

3. Experimental analysis

In the course of the study, a file-by-file processing methodology was applied to the benchmark set. Each file was analyzed sequentially using a static analysis tool. In the event of errors indicating the absence of required modules or files, an attempt was made to locate the missing components within the benchmark collection itself.

If the necessary files could not be found, the case was labeled as an error, and a stub module with a required interface was created. These stub modules were placed in the same directory as the original file to ensure correct path resolution during inclusion. For each recurring type of error, a generalized description was compiled, including the error category and the recommended correction (see Table 1). Descriptions of certain errors related to implementation specifics were supplemented with detailed explanations.

After applying the corrections, each file was re-analyzed using the same tool to verify that the error had been resolved and that no further diagnostic messages were issued by the analyzer. No time limits were set for the static analysis tool.

The main contribution of this paper consists of two key aspects. First, a comprehensive comparative evaluation of existing static analysis tools was conducted in the context of digital circuit verification tasks. Second, a number of errors were identified and corrected within a widely used benchmark collection, thereby improving its overall quality and suitability for use in both academic and practical applications.

As part of the conducted research to analyze the applicability of SVAN for open RTL benchmarks, the tool was tested on the open-source `hdl-benchmarks` collection. This collection includes the following sets of benchmarks:

1) ICCAD-2015 CAD Contest benchmark suite;
2) ISCAS'85 benchmarks [25];
3) ISCAS'89 benchmarks [26];
4) IWLS'2005 benchmarks [17];
5) LGSynth'1991 benchmarks [27];
6) Quartus University Interface Program (QUIP) benchmarks [18];
7) Texas-97 benchmarks [28];
8) VCEGAR benchmarks [29];
9) Verilog2SMV benchmarks [22];

These benchmarks were widely used for the verification of HDL design and synthesis tools [23-24]. However, it has not yet been analyzed using static analysis methods. These projects encompass a broad spectrum of algorithms and functionalities, which served as the decisive factor in their selection as a suitable object for comprehensive analysis. Such a wide-ranging representation ensures that the tool's capabilities are tested under conditions that closely mimic real-world hardware design scenarios, thereby providing a robust assessment of its effectiveness and reliability.

Results of experimental evaluation of analyzed tools (with default running options) are shown in Table 1. On the first column error types are described. Last columns are named by the related static analysis tools: SVAN (ISP RAS), VCS (Cadence) V-2023.12 and Verilator 4.028 2020-02-06 respectively.

Table 1. Errors are detected by Static Analysis Tools.

Error type	SVAN	VCS	Verilator
Unknown macros or compiler directives detected	36	8	134
Expected expression	35	0	0
Unknown module	290	236	97
No implicit conversion; explicit conversion exists but casting is missing	2	0	0
Unable to resolve hierarchical path	1	0	0
Module redefinition	4	0	0
No such file or directory	26	77	79
Invalid delay value expression	2	0	0
Size requires a constant range	3	0	0
Too many arguments passed	1	0	0
Incompatible bit widths	1	0	0
Unknown system name	2	0	0
Missing identifier	5	0	0
Too many connections specified for port instantiation	2	2	0
Other	23	0	24
Total	*433*	*323*	*334*

The errors classified as "*Unknown macros or compiler directives detected*" are essentially a consequence of the absence of the source code referencing macros or directives that are not defined within the analyzed files or their associated include paths. Below is an example of the error containing code and related diagnostics:

```
unknown macro or compiler directive '`RDY'
            if (ACK == `RDY)
```

The "*Unknown module*" error indicates that the code instantiates a module whose declaration is not present in the example being analyzed by the tool.

The "*No such file or directory*" error occurs when the code attempts to include a header or other file that does not exist in the directory where the analyzed code is located. This type of error is the second most frequently encountered among those identified. Its presence in the benchmarks may be attributed to inaccuracies introduced during the aggregation of the original projects into the `hdl-`

`benchmarks` collection. However, configuring the tool execution flow does not provide an opportunity to rectify such errors, as it is not feasible to recover the missing components and reconstruct the projects outside the scope of this collection.

Table 2 presents the solutions proposed as part of the measures taken to rectify errors present in the open-source benchmarks. In particular, to address errors related to missing modules, functions, or macro definitions, a search was conducted for the corresponding file within the project directories. If the file was not found, a stub file implementing the required module, function, or macro was created accordingly. Additionally, to resolve the issue of missing explicit type casting, it was introduced using a system task.

During the experimental testing, it was determined that the SVAN tool demonstrates significant superiority over both the commercial Synopsys VCS and the open-source Verilator tool across several key metrics. In particular, the number of classes of errors detected by SVAN (15) was found to be 73% broader than that both of VCS (4) and Verilator (4). Additionally, the total number of errors identified by SVAN (433) in the analyzed RTL model modules exceeds the corresponding metric for VCS (323) by 25% and for Verilator (332) by 23% (see “Total” row in Table 1).

This difference can be attributed to several factors.

Table 2. Errors and Suggested Solutions.

Error type	Suggested correction
Unknown macros or compiler directives detected	Check the project directory to see whether the required file exists. If it does, import it into the module. If not, create a stub file in the project directory and import it into the module.
Expected expression	
Unknown module	
No such file or directory	
No implicit conversion; explicit conversion exists but casting is missing	Add an explicit cast using `$sformatf("%s", in_file)`
Unable to resolve hierarchical path	In the included file, initialize the used parameters and assign them appropriate values
Module redefinition	Review all included files to determine whether the same file is being included twice through different paths into the module under investigation. Then adjust the file inclusion order to prevent module redeclaration
Invalid delay value expression	Add a file containing the defined delay value expressions
Size requires a constant range	Include a `.vh` file that contains the required constant range definitions
Too many arguments passed	Modify both the function definition and its usage within the module so that no more arguments are passed than are required
Unknown system name	Add a stub file implementing a function with the specified name
Missing identifier	Declare the missing identifier
Too many connections specified for port instantiation	Correct the instance declaration so that no more port connections are provided than are necessary

3.1 Functional limitations of Synopsys VCS and Verilator

The VCS tool terminates its analysis upon encountering the first error of a specific category. For instance, if an error arises due to a missing included file or directory, the tool halts further error detection not only within that category but also across all other categories, even though additional errors may still exist in the code. There are no any options allowing to continue further errors detection, which leads to significant restrictions the comprehensiveness of the analysis.

Verilator does not correctly handle the `‘include <path-to-file>’` directive (even when using the additional command-line argument `+incdir+<path-to-dir>`) which is used to include external files in module code. As a result, the tool fails to locate included files, even when they exist in the same directory as the module being analyzed. This limitation leads to a large number of false negatives, as Verilator generates errors related to “missing” files that are actually accessible during compilation.

By contrast, SVAN continues its analysis regardless of the presence of initial errors, ensuring a more thorough and exhaustive examination of the codebase. This capability allows SVAN to identify a broader spectrum of errors, thereby providing a more complete picture of the design's integrity and potential vulnerabilities.

3.2 Granularity of error classification

SVAN provides a more detailed classification of errors compared to both VCS and Verilator. For instance, errors related to invalid delay value expressions and the absence of macro or compiler directive definitions are grouped under a single marker in VCS, labeled as "*Error-[UM] Undefined macro*".

Verilator assigns a broad category of "*Syntax error*" to multiple distinct error types identified by SVAN, such as missing identifiers, unknown system names, and passing too many arguments to a function.

This broad categorization can obscure the specific nature of the issues, making it more challenging for developers to identify and address the root causes effectively.

In contrast, SVAN distinguishes between these categories of errors, enabling more precise diagnostics. For example, issues like missing predefined delay expressions, undefined macros, unresolved hierarchical paths, and incorrect argument counts are flagged as independent errors. This approach allows users to pinpoint the exact nature of each issue, facilitating more efficient debugging and resolution. By providing clearer and more granular feedback, SVAN ensures that developers receive actionable insights into the specific problems within their code, ultimately enhancing the overall quality and reliability of the design process.

3.3 Absence of detection for certain error categories

Furthermore, the analysis revealed significant limitations in both Synopsys VCS and Verilator.

Synopsys VCS tool does not detect certain categories of errors, such as the absence of implicit type casting. This limitation can result in potentially problematic code sections going unnoticed, thereby increasing the risk of errors during synthesis or simulation stages. For instance, issues related to incompatible data types or missing explicit type conversions may remain undetected, leading to unpredictable behavior or functional failures in the hardware design.

Verilator has an even narrower range of detectable errors. It cannot identify issues such as unresolved hierarchical paths, module redefinitions, or many other errors that could impact the correctness of the system.

On the contrary, SVAN is capable of identifying such non-obvious errors due to its inclusion of detectors specifically designed to verify the correct usage of data types. These detectors explicitly check for cases where implicit type casting is expected but absent, ensuring that all type-related issues are flagged and addressed. By incorporating these advanced checks, SVAN provides a more thorough analysis of the code, reducing the likelihood of oversight and enhancing the overall robustness of the design process.

During experimental testing, it was also found that 5% of the analyzed RTL models were incorrectly classified as containing errors by SVAN and Verilator, whereas the Synopsys VCS tool marked them as correct and they were actually correct. They were marked as "*Other*" in the Table 1.

Those issues arise due to the specific characteristics of code analysis performed in accordance with the Verilog IEEE 1364-2005 and SystemVerilog IEEE 1800-2023 standards. When verifying benchmarks containing mixed code written in both languages, valid Verilog code did not meet the stricter requirements of the SystemVerilog standard. For example, a variable named `do` was recognized as a reserved keyword introduced in the SystemVerilog standard. As a result, such models were flagged as erroneous, even though they were actually compliant with the Verilog standard.

The broader spectrum of detectable errors, increased level of analysis detail, and absence of the limitations of VCS and Verilator make SVAN a more effective tool for the static verification of RTL models.

Table 3 shows the errors distribution on the selected benchmarks. The analysis reveals that the IWLS-05 benchmark contains the highest number of errors among the evaluated datasets. Specifically, SVAN, VCS, and Verilator detected 214, 214, and 212 errors (see "*IWLS-05*" row in Table 3) respectively, in this benchmark, making it the most error-prone dataset in absolute terms. A significant proportion of these errors – 77% (165 out of 214) – are attributed to the absence of a file containing the module whose instance is instantiated in the analyzed file. This issue represents the most frequently encountered error type in the IWLS-05 benchmark.

Another notable dataset is the QUIP benchmark, which was released as part of Altera's educational program. While QUIP is not the largest benchmark in terms of error count, it exhibits a relatively high error rate, with 99 errors detected (see "QUIP" row in Table 3). The higher error frequency in QUIP can be attributed to its lesser degree of debugging compared to other benchmarks. Similar to IWLS-05, the most common error in QUIP is the absence of a module corresponding to an instantiated instance. Additionally, the minor variability in error detection rates among the tools suggests that QUIP presents specific challenges for verification tools.

Table 3. Errors to Benchmarks distribution.

Benchmark	SVAN	VCS	Verilator
Verilog2SMV	1	1	1
VCEGAR	5	0	5
TEXAS-97	17	17	17
QUIP	99	99	97
IWLS-05	214	214	212
ISCAS89	4	4	4

The benchmark with the fewest errors, based on the results of the empirical study, is Verilog2SMV (see "Verilog2SMV" row in Table 3). This may be attributed to the fact that this benchmark serves as a test suite specifically developed for testing the eponymous tool by the Bruno Kessler Foundation [30].

Additionally, only four errors of the type "*Unknown Module*" were detected by all tested static analysis tools on the ISCAS-89 benchmark. The ISCAS-89 benchmark was originally distributed on tape to participants of the Special Session on Sequential Test Generation at the International Symposium on Circuits and Systems in May 1989 and is partially characterized in [26]. This is the reason for the number of errors contained in this benchmark being minimal and representing only one class of errors outlined in Table 1.

Overall, the tools SVAN and VCS demonstrate robust performance across all benchmarks, consistently identifying the highest number of errors. Their consistent results highlight their reliability in detecting issues, even in less refined datasets like QUIP.

In terms of tool performance, SVAN and VCS demonstrate nearly identical results across all benchmarks, consistently detecting the same number of errors. The primary discrepancy lies in the complete absence of error detection by VCS within the VCEGAR benchmark, where SVAN demonstrates superior performance by identifying errors, matching the results achieved by Verilator. This highlights the robustness of SVAN in handling the complexities of the VCEGAR dataset, further solidifying its position as a leading tool in error detection across diverse benchmarks.

4. Conclusions and final remarks

Based on the empirical study, a number of unique characteristics of SVAN were identified, setting it apart from existing solutions for the verification of hardware descriptions written in Verilog and SystemVerilog. The key advantage of SVAN compared to Synopsys VCS and Verilator is high analysis accuracy. SVAN's ability to provide more granular and precise diagnostics ensures that

even subtle issues are identified and properly categorized, reducing the likelihood of undetected errors that could compromise the design process.

Despite its significant advantages, the SVAN tool has certain limitations that must be taken into account when using it. One of the key drawbacks is its insufficient flexibility in handling RTL models are written in two hardware description languages – both Verilog and SystemVerilog. This limitation highlights the need for further improvements in SVAN's ability to handle mixed-language designs and to differentiate between language-specific constructs more effectively. While SVAN's strict adherence to SystemVerilog standards ensures high accuracy in many cases, it can also lead to false positives when analyzing legacy Verilog code or designs that combine both languages. Addressing this challenge will be critical for enhancing the tool's compatibility and usability across a wider range of hardware design projects.

The results presented in this work are valuable both in terms of comparing the developed SVAN tool with proprietary counterparts and in a broader context. On one hand, the study has a scientific focus, involving the analysis and comparison of functional capabilities among existing static analysis tools. On the other hand, it can be regarded as a technical report reflecting the current state and practical capabilities of the SVAN tool.

The further development of this research includes a broader comparison of the static analysis tool SVAN with existing commercial and non-commercial tools, including an assessment of performance and functional capabilities. This will be the subject of future studies and will allow for a deeper evaluation of the proposed method's potential under real-world application conditions.

References

[1]. IEEE SA. IEEE Standard for Verilog Hardware Description Language [Online]. Available: https://standards.ieee.org/ieee/1364/3641/ (accessed 2025, Mar. 27).

[2]. IEEE SA. IEEE Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language [Online]. Available: https://standards.ieee.org/ieee/1800/7743/ (accessed 2025, Mar. 27).

[3]. NVDLA Project, "NVIDIA Deep Learning Accelerator (NVDLA) – Open-Source Inference Accelerator", GitHub repository, 2017-2023. [Online]. Available: https://github.com/nvdla (accessed 2025, Mar. 27).

[4]. GOST R 71207-2024, "Information Protection. Secure Software Development. Static Code Analysis. General Requirements", Moscow: Standartinform, 2024. [Online]. Available: https://protect.gost.ru/v.aspx?control=7&id=257752 (accessed 2025, Mar. 27).

[5]. Synopsys, "VCS® Functional Verification Solution", Synopsys Inc., 2023. [Online]. Available: https://www.synopsys.com/verification/simulation/vcs.html (accessed 2025, Mar. 27).

[6]. Synopsys, "SpyGlass: Early Design Analysis Tools for SoCs", Synopsys Inc., 2025. [Online]. Available: https://www.synopsys.com/verification/static-and-formal-verification/spyglass.html (accessed 2025, Jun.12).

[7]. Cadence Design Systems, "Formal and Static Verification Solutions", Cadence Inc., 2023. [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification.html (accessed 2025, Mar. 27).

[8]. Siemens EDA (2025, Mar. 27). Questa Verification Solutions, Siemens Digital Industries Software, 2023. [Online]. Available: https://eda.sw.siemens.com/en-US/ic/questa/ (accessed 2025, Mar. 27).

[9]. M. Popoloski, slang: SystemVerilog Language Services, version 8.0, 2015-2025. [Online]. Available: https://github.com/MikePopoloski/slang (accessed 2025, Mar. 27).

[10]. Verilator. Verilator – a fast Verilog/SystemVerilog simulator. GitHub repository, 2003-2024. [Online]. Available: https://github.com/verilator/verilator (accessed 2025, Mar. 27).

[11]. D. Alance. svlint – SystemVerilog linter. GitHub repository 2020-2024. (2025, Mar. 27). [Online]. Available: https://github.com/dalance/svlint (accessed 2025, Mar. 27).

[12]. CHIPS Alliance, Surelog v1.84 – SystemVerilog 2017 Toolchain, 2024. [Online]. Available: https://github.com/chipsalliance/Surelog (accessed 2025, Mar. 27).

[13]. CHIPS Alliance. Verible – Suite of SystemVerilog developer tools. GitHub repository, 2019-2024. [Online]. Available: https://github.com/chipsalliance/verible (accessed 2025, Mar. 27).

[14]. T. Pecenka, L. Sekanina, and Z. Kotasek, "Evolution of synthetic RTL benchmark circuits with predefined testability", ACM Trans. Des. Autom. Electron. Syst., vol. 13, no. 3, pp. 1–21, Jul. 2008.

[15]. F. Yuan, “Design and Test for Timing Uncertainty in VLSI Circuits”, Ph.D. dissertation, Chinese University of Hong Kong, Hong Kong, 2012.
[16]. Ya. A. Churkin, R. A. Buchatskiy, K. N. Kitaev, A. G. Volokhov, E. V. Dolgodvorov, A. S. Kamkin, A. M. Kotsynyak, and D. O. Samovarov, “Static Analysis System for SystemVerilog Hardware Description Language”, Proceedings of the ISP RAS, vol. 37, no. 1, pp. 7–40, 2025.
[17]. International Workshop on Logic & Synthesis (IWLS), “IWLS 2005 Benchmark Suite”, 2005. [Online]. Available: https://iwls.org/iwls2005/benchmarks.html (accessed 2025, Mar. 27).
[18]. N. Isaac, “QUIP Toolkit Benchmarks (v9.0)”, ECE496 GitHub repository, 2020. [Online]. Available: https://github.com/neilisaac/ece496/tree/master/reference/quip_toolkit-9.0/benchmarks (accessed 2025, Mar. 27).
[19]. hdl-benchmarks – Collection of digital hardware modules & projects (benchmarks), GitHub repository, 2019-2023. [Online]. Available: https://github.com/ispras/hdl-benchmarks (accessed 2025, Apr. 06).
[20]. Freescale Semiconductor, Verilog HDL Coding: Semiconductor Reuse Standard. Freescale Semiconductor, Inc., 2005.
[21]. M. Taylor and Bespoke Silicon Group. BSG SystemVerilog Coding Standards, University of Washington, 2023.
[22]. C. Wolf, “PicoRV32 – A Size-Optimized RISC-V CPU Core”, YosysHQ GitHub repository, 2015-2023. [Online]. Available: https://github.com/YosysHQ/picorv32 (accessed 2025, Mar. 27).
[23]. OpenHW Group, “CVA6 – An Open-Source 64-bit RISC-V CPU Core”, GitHub repository, 2019-2023. [Online]. Available: https://github.com/openhwgroup/cva6 (accessed 2025, Mar. 27).
[24]. lowRISC, “OpenTitan – Open Source Silicon Root of Trust”, GitHub repository, 2018-2023. [Online]. Available: https://github.com/lowRISC/opentitan (accessed 2025, Mar. 27).
[25]. M. Hansen, H. Yalcin, J. Hayes, “Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering”, IEEE Design & Test of Computers, vol. 16, no. 3, pp. 72-80, 1999.
[26]. F. Brglez, D. Bryan, K. Kozminski, “Combinational Profiles of Sequential Benchmark Circuits”, Proc. IEEE Int. Symposium on Circuits and Systems, pp. 1929-1934, May 1989.
[27]. Microelectronics Center in North Carolina. LGSynth91 benchmarks. [Online]. Available: https://ddd.fit.cvut.cz/www/prj/Benchmarks/LGSynth91.7z (accessed 2025, Jun. 14).
[28]. Department of Electrical and Computer Engineering at the University of Texas. Texas-97 benchmarks. [Online]. Available: https://ptolemy.berkeley.edu/projects/embedded/research/vis/texas-97 (accessed 2025, Jun. 14).
[29]. University of Oxford. VCEGAR benchmarks. [Online]. Available: http://www.cprover.org/hardware/benchmarks/vcegar-benchmarks.tgz (accessed 2025, Jun. 14).
[30]. FBK, “Verilog2SMV”, Fondazione Bruno Kessler (FBK). [Online]. Available: https://es-static.fbk.eu/tools/verilog2smv/ (accessed 2025, Mar. 29).

Информация об авторах / Information about authors

София М. Панова является лаборантом-исследователем отдела Технологий Программирования Института системного программирования им. В.П. Иванникова Российской академии наук (ИСП РАН). Область научных интересов: цифровая аппаратура, статический анализ, функциональная верификация.

Sophia M. Panova – Laboratory assistant at the Software Engineering Department of the Ivannikov Institute for System Programming of the Russian Academy of Sciences (ISP RAS). Her research interests include digital hardware design, static analysis, and functional verification.

Сергей А. СМОЛОВ является научным сотрудником отдела Технологий Программирования Института системного программирования им. В.П. Иванникова Российской академии наук (ИСП РАН), старший научный сотрудник научной лаборатории «Гетерогенные компьютерные системы» РЭУ им. Г.В. Плеханова. Область научных интересов: автоматизация проектирования цифровой аппаратуры, верификация и тестирование.

Sergey A. SMOLOV – Researcher at the Software Engineering Department of Ivannikov Institute for System Programming of the Russian Academy of Sciences (ISP RAS), senior researcher at the

Heterogeneous Computing Systems research lab of Plekhanov RUE. His research interests include digital hardware design automation, verification and testing.

Марина М. ВОЛКОВА является лаборантом отдела Технологий Программирования Института системного программирования им. В.П. Иванникова Российской академии наук (ИСП РАН). Область научных интересов: машинное обучение, интеллектуальный анализ данных, методы статического анализа и верификации.

Marina M. VOLKOVA – Laboratory assistant at the Software Engineering Department of Ivannikov Institute for System Programming of the Russian Academy of Sciences (ISP RAS). Her research interests include machine learning, data analysis, methods of static analysis and verification.

DOI: 10.15514/ISPRAS-2025-37(5)-11

Designing Refactoring Tool for Object-Oriented Code Based on Metrics

[1] *A.O. Korznikov, ORCID: 0009-0006-3941-9214 <artemkorz@mail.ru>*
[2,3] *N.N. Datsun, ORCID: 0000-0001-8560-7036 <nndatsun@inbox.ru>*

[1] *Perm State University,*
15, Bukireva st., Perm, 614068, Russia.

[2] *HSE University, Perm,*
38, Studencheskaya st., Perm, 614070, Russia.

[3] *PSHPU, Perm,*
24, Sibirskaya st., Perm, 614990, Russia.

Abstract. Currently, the information technologies industry is a leader in growth rate among the main economic sectors. However, the most important components of the development process, such as estimation and refactoring of program products, still remain without generic tools. Therefore, our main goal is to design a mean of unified modification and formal evaluation for code in object-oriented programming languages. We use refactoring patterns to define code modifications, and code metrics calculation to assess its characteristics. Our tool should help developers to make decisions connected with code quality and its modification necessity, automatize that change. Actually, it may be used in organizations and educational institutions. We have developed a domain specific language to unify the specification of object-oriented languages. Furthermore, a research prototype of the tool has been created. 3 object-oriented languages descriptions and 6 diverse refactoring patterns have been developed to demonstrate capabilities of the approach.

Keywords: refactoring; domain specific language; code metrics calculation; object-oriented language; refactoring patterns.

For citation: Korznikov A.O., Datsun N.N. Designing Refactoring Tool for Object-Oriented Code Based on Metrics. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 143-156. DOI: 10.15514/ISPRAS-2025-37(5)-11.

Проектирование инструмента для рефакторинга объектно-ориентированного кода с использованием расчета метрик

[1] *А.О. Корзников, ORCID: 0009-0006-3941-9214 <artemkorz@mail.ru>*
[2,3] *Н.Н. Дацун, ORCID: 0000-0001-8560-7036 <nndatsun@inbox.ru>*

[1] *Пермский государственный национальный исследовательский университет, Россия, 614068, г. Пермь, ул. Букирева, д. 15.*

[2] *НИУ ВШЭ, Пермь, Россия, 614070, г. Пермь, ул. Студенческая, д. 38.*

[3] *Пермский государственный гуманитарно-педагогический университет, Россия, 614990, г. Пермь, ул. Сибирская, д. 24.*

Аннотация. На данный момент отрасль информационных технологий (ИТ) занимает лидирующую позицию по темпам роста среди основных отраслей экономики. Однако, не существует универсальных и стандартизованных инструментов для важнейших компонентов процесса разработки: оценки и рефакторинга программных продуктов. Поэтому нашей основной целью является проектирование средства для унифицированного изменения и формальной оценки кода на объектно-ориентированном языке программирования. Для описания изменений кода используются шаблоны рефакторинга, для оценки его характеристик – расчет метрик кода. Целью нашего инструмента является помощь программистам в принятии решений, связанных с качеством кода и необходимостью внесения изменений в код, автоматизация этих изменений. Приложение может использоваться в организациях и образовательных учреждениях. Был разработан предметно-ориентированный язык, чтобы унифицировать описание объектно-ориентированных языков. Кроме того, был создан исследовательский прототип инструмента. Для демонстрации возможностей предложенного подхода были созданы 3 описания объектно-ориентированных языков и 6 различных шаблонов рефакторинга.

Ключевые слова: рефакторинг; предметно-ориентированный язык; расчет метрик кода; объектно-ориентированный язык; шаблоны рефакторинга.

Для цитирования: Корзников А.О., Дацун Н.Н. Проектирование инструмента для рефакторинга объектно-ориентированного кода с использованием расчета метрик. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 143–156 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)–11.

1. Introduction

At present a rapid rise of IT industry occurs. Despite this, a number of significant issues that inhibit the effective work and development of organizations are detected. Firstly, there are no standardized tools for evaluating computer programs. Secondly, refactoring is a key and integral part of software maintenance, but there is no common tool for it. Thirdly, a rate of technology substitution is high, and it is vital for organizations to use effective ones to stay competitive.

Metrics can be considered as a tool for formal evaluation of code characteristics. However, the recent researches show that applications based on metrics have a low true positive rate [1]. The authors believe that lack of an actual context is the main reason [2]. In our work, the metrics are interpreted by a programmer, who uses our tool. Developer as an expert assesses a code quality using personal professional experience and recommended limits for metrics values provided by the tool. Moreover, it is a useful practice to compare the characteristics of different program projects.

Refactoring is a mean of the uncontrollable growth prevention of a program code length, number of errors hidden, and design issues (technical debts). Therefore, the significance of refactoring cannot be underestimated, but its implementation is connected with high complexity and time cost, especially when changing the whole project [3]. Modern IDEs include means of metrics calculation as well as rapid global refactoring while coding (floss refactoring) [4: 163]. However, they may be inconvenient in some cases due to massiveness. Furthermore, IDEs focus on supporting a small number of programming languages and specific refactoring operations. In addition, researches show

that many developers are cautious about these tools and prefer to perform refactoring manually [4: 163, 5: 4, 6: 1]. Our proposed approach and a research prototype based on it are independent of object-oriented programming (OOP) language.

Actually, different organizations use various versions and extensions for programming languages, some of them develop the own languages to solve particular tasks in a specific domain. On the one hand, it helps to accelerate the development and simplify understanding, but on the other hand, refactoring and metrics calculation may become complicated. Thus, extra financial resources are required to develop appropriate tools for automatic execution of those actions.

To implement the tool for refactoring object-oriented code using metrics calculation, the following tasks must to be solved to:

- analyze requirements;
- analyze existing software;
- design the program tool;
- design a domain specific language (DSL);
- implement the tool and test it.

2. Requirements

To solve these problems, a tool independent of a specific object-oriented languages set should be developed. It must:

- provide means of a unified description of various languages and refactoring operations;
- use terms of the procedural and OOP paradigms to calculate the respective metrics;
- describe languages and refactoring operations in a similar way to simplify the tasks of programmer;
- apply refactoring to the whole project or its individual physical (files) and logical (hierarchies of classes) parts;
- allow developers to evaluate the formal characteristics of program code, compare metrics of refactored code with initial values.

3. Related works

3.1 Technical debt and code smells

In practice, developers often use refactoring to remove “technical debt”, particularly “code smells” [7]. Technical debt means a decrease of code quality in its development [1]. In the long term, it leads to such serious consequences as an increase in cost of defects correction, further development and making changes [1, 8]. Code smells are the most studied and recognizable type of technical debt related to design problems [8]. The term is firmly entrenched in the context of refactoring and combines the problems encountered in object-oriented code [1].

Code metrics can also be used to determine this kind of drawback [1], especially Chidamber and Kemerer set is often used. According to various works, metrics in the context of refactoring are applied for identification of low-quality code parts [9], comparing a source code with refactored one [10] and estimating a cost of refactoring application [11].

3.2 Refactoring tools

The study of current refactoring tools is presented in a systematic mapping study [12]. According to the data obtained, there is a set of refactoring applications that recommend changes, perform refactoring and detect it [12]. The most commonly considered language is Java, and there are also

tools working with code in C, JavaScript, and various DSL (e.g., CSS). The denoted tools are focused on applying specific refactoring operations within a certain set of languages.

In the review [13], Eclipse, Xcode are named among the popular refactoring tools. Moreover, some of the applications solve similar issues, such as refactoring tests, performed using DARTS [10] and B-refactoring [12]. Other systems are highly specialized and work with a particular domain: RIdiom automatically replaces all code fragments that do not correspond to Python idioms [14]; ReSwither modifies structure of a switch operator in Java [15]; Android Studio plugin works with energy efficiency [16]. Besides, a lot of the presented tools are either unavailable in Russia or are not free: Synchronizer, Asyncdroid, Xll and others [12].

3.3 Tools for code metrics calculation

Systematic mapping studies (SMS) [17-19] examine the possibilities of tools for automated metric calculation. It is worth noting a similar study, the authors of which indicated that SourceMeter and Metrics are most often used to calculate program metrics; PMD and JDeodorant are commonly used for detection and removing bad smells; JDeodorant and Eclipse are most frequently applied for refactoring [20: 929]. However, the applications for refactoring, metrics calculation, and tools proposed by IDEs, which are discussed in these papers, are not universal.

4. Describing approach and designing tool

4.1 Framework

We propose a refactoring approach described in Fig. 1. An iteration of the refactoring loop [21] requires to identify (step 3) code fragments, to recommend (steps 2, 5) metrics comparison evaluation, and to apply (steps 4, 6) code modifications. A green outline shows a preparatory step performed by the programmer manually. The blue frames depict steps performing semi-automatically, when a decision is made by user. Other steps are done automatically. Thus, in refactoring loop we suggest a user to choose the refactoring pattern and to decide whether to apply the modification using results of the metrics comparison.

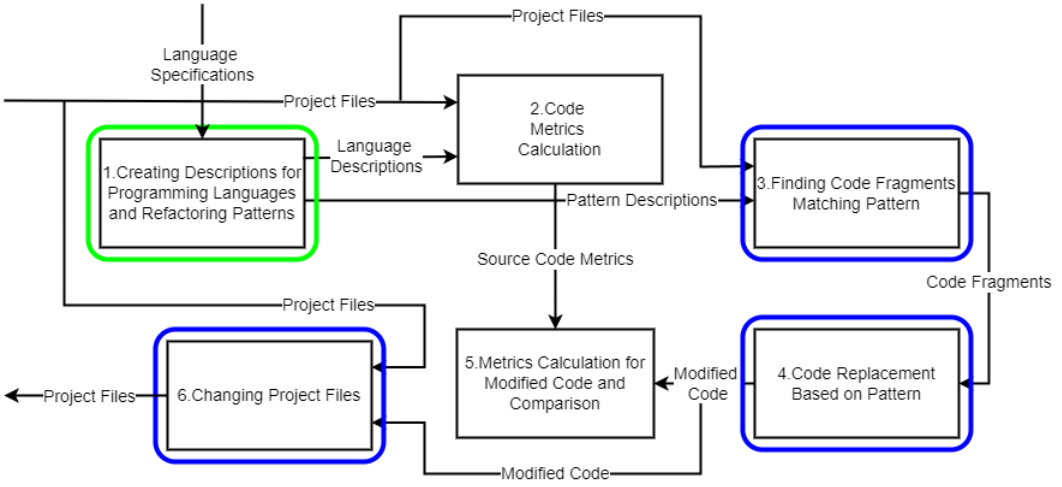


Fig. 1. Proposed approach framework.

4.2 Designing tool

The main purpose of our tool is to obtain values of the formal code metrics, perform refactoring, and compare calculated parameters. As the determined indicators, Chidamber and Kemerer, Lorenz

and Kidd metrics sets were selected. Additionally, Halstead metrics, lines of code (LOC) and program style evaluation which are independent of a paradigm, were included in the set used.

The tool consists of 4 modules (Fig. 2): (1) an analyzer, (2) metrics calculator, (3) explanation and (4) refactoring units. The analyzer includes lexical, syntactic and semantic code analysis. As we develop a generic tool, only the general semantics of object-oriented languages is considered and the rest of it must be specified in a particular language description. The metrics calculation is performed both for individual elements of object-oriented languages and for a whole project by calculating average values. To explain the results, a dictionary that includes namespaces and classes is used, as well as a comparison between the calculated metrics values and numbers recommended by their authors.

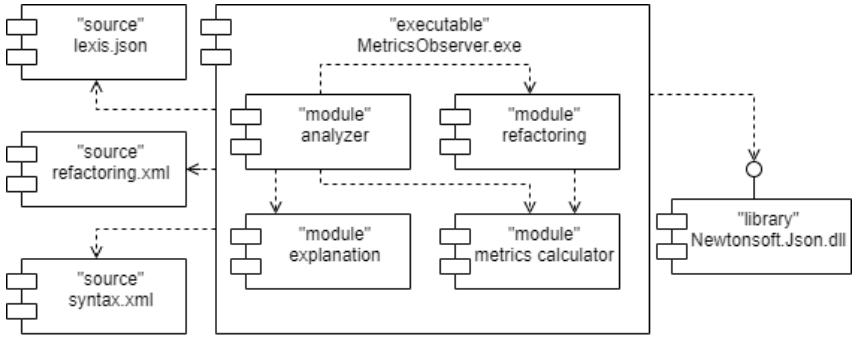


Fig. 2. Tool architecture.

4.3 Designing DSL

To provide an opportunity of analyzing the code in various languages, a textual DSL was developed. It allows a programmer to describe lexis and syntax of the OOP languages. The language is based on terms such as class, namespace, operator, and operand. BNF was defined for the DSL.

A pattern description method was selected to implement refactoring [22] and code parts. Therefore, it provides a possibility to work similarly with the language syntax and the refactoring pattern. The refactoring pattern structure that consists of 4 parts (Fig. 3): variables, search, replace, and references section was proposed. Firstly, a list of entities required and their initial values is described. Secondly, elements of a language and its context must be defined as the syntax is. That definition could be placed separately as a code fragment description.

<Refactorings> ::= <Pattern> {<SubPattern> | <Fragment>}
<SubPattern> ::= <Pattern>
<Pattern> ::= “<Refactoring xsi:type="Pattern">” <PatternBody> <Next> “</Refactoring>”
<PatternBody> ::= <Name> <Variables> <Search> <Replace>
<Name> ::= “<Name>” <String> “</Name>”
<Variables> ::= “<Variables>” <PosEntity> {<PosEntity> | <VariableEntity>} “</Variables>”
<PosEntity> ::= <PositionsVar> | <PositionVar>
<PositionsVar> ::= “<Variable xsi:type="Position">” <String> “</Variable>”
<Search> ::= “<Search>” <DescriptionBody> “</Search>”
<DescriptionBody> ::= <Instruction> {<Instruction>}
<Replace> ::= “<Replace>” {<Insert>} “</Replace>”
<Insert> ::= “<Entity xsi:type="" <InsertType> “">” <Position> [<Description>] “</Entity>”
<Next> ::= “<Next>” <Refactor> {<Refactor>} “</Next>”
<Refactor> ::= “<Refactor>” <String> <Arguments> “</Refactor>”
<Fragment> ::= “<Refactoring xsi:type="Fragment">” <FragmentBody> “</Refactoring>”
<FragmentBody> ::= <Name> <Variables> <Description>

Fig. 3. Fragment of BNF for refactoring pattern structure.

Subsequently, the replacement part requires position of the code fragment to be defined. The location is set as a tuple of a first character position and symbols amount. It can be found by the code

description. The code replacement requires the position and a type of change: adding or modification. Besides, it would be inevitable to perform the search again in some cases. Thus, the last part of the pattern is used for referring other ones and share information accumulated.

Visualization of replacements made is also a significant aspect. We decided to highlight the code fragments related to the first entity described. As a result, the corresponding fragments of a source and modified code are colored identically. It allowed highlighting the refactored code parts automatically. Additionally, the developer can assign a color independently. It might be useful when creating additional files while refactoring.

5. Implementation and testing

The implemented DSL was based on xml as it is conveniently serializable. It was necessary to define descriptions of OOP languages for testing. For this reason, the lexis and syntax were defined for subsets of C# 7.3, Java SE 8, and C++11 using the DSL. The main purpose of those definitions is to provide an opportunity to test the research prototype developed. Moreover, some language details are not significant for metrics calculation, because of that a complete description is not required (e.g., C++ pragma instructions).

5.1 Testing refactoring patterns

We have implemented 6 refactoring instances and they have been tested using a code in C#. Fig. 4-10 show the source code on the left and the refactored one on the right.

Fig. 4 demonstrates a sort of imports. It is performed sequentially for each word in the compared strings and started using the reserved command for the list. The corresponding lines are automatically highlighted with colors. The corresponding declarations are created.

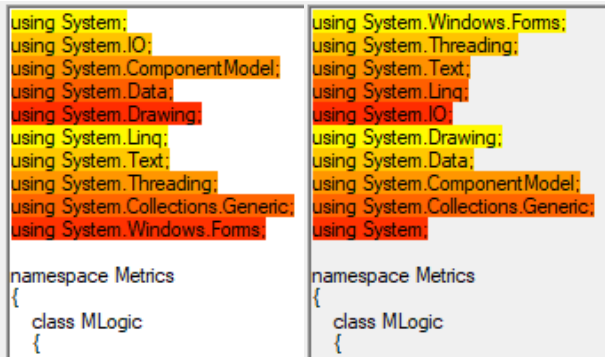
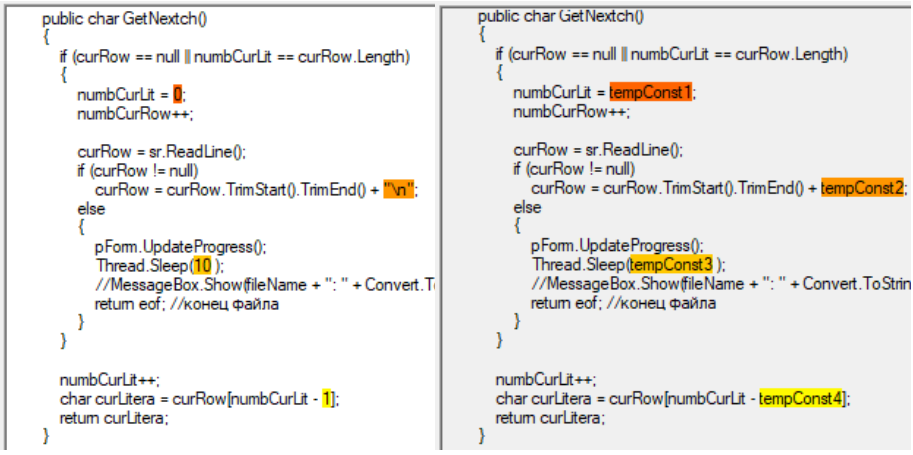


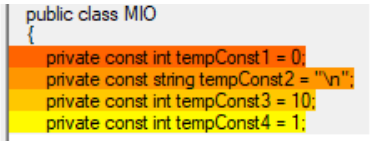
Fig. 4. Sorting imports.

Fig. 5 (*a*) shows an example of moving literals from class methods to constants using the stated name. Actually, they are numbered automatically and the same values are considered as the same constant. Code fragments are skipped until an operand that is a literal is encountered. The unique values of variables are stored using sets, whereas positions and values are contained with stacks. The corresponding changes in code have the identical color, and declarations of the constants are highlighted with the last change color (Fig. 5, *b*). Fig. 6 illustrates an example with C++ code.

Fig. 7 depicts an example of the following pattern. It allows a programmer to move the code located between comments of a certain type, defined by a regular expression. For that reason, a method with described modifiers and name is created. Although the comments shown are convenient for processing with that pattern, they are practically meaningless and do not contribute to documentation of an application. After performing this type of pattern, the documentation refactoring proposed in [23] should be performed additionally.



(a)



(b)

Fig. 5. Transforming literals into constants (C#):
a) source and refactored code; b) declarations of constants in refactored code.

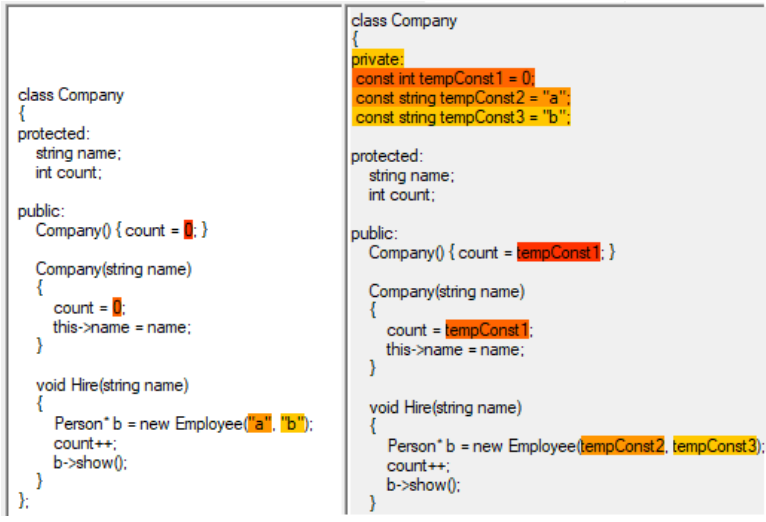


Fig. 6. Transforming literals into constants (C++).

Fig. 8 demonstrates an example of using a pattern when a developer requires creating a corresponding property for a public field.

Fig. 9 (*a*) shows an example of refactoring that extracts an interface from a class into new separated file. After public methods are found, their signatures are transferred to the extracted interface in the created file (Fig 8, *b*). Furthermore, inheritance code related to the interface is added to the class. The color design is performed manually for inheritance, as it is a new code fragment added to the existing set.

Fig. 10 depicts an example of move method refactoring that is often used in practice [24]. The idea is to find a calls number of the specified method for each class and move it to the one with the highest value.

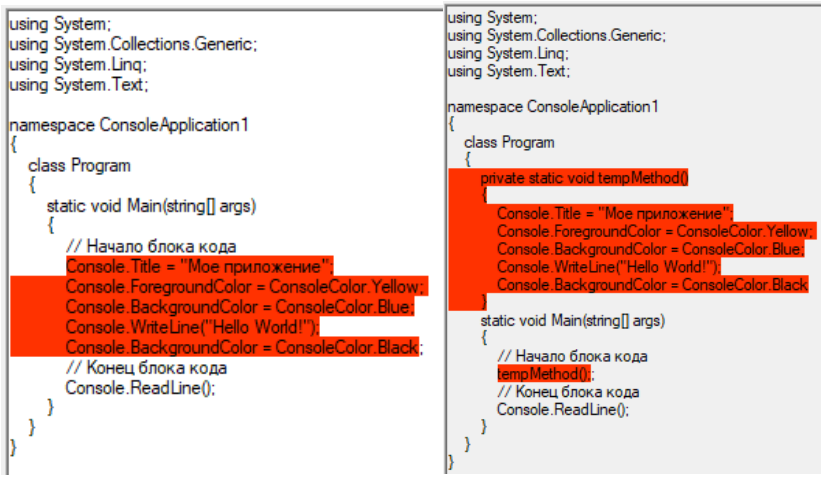


Fig. 7. Moving code into method using comments.

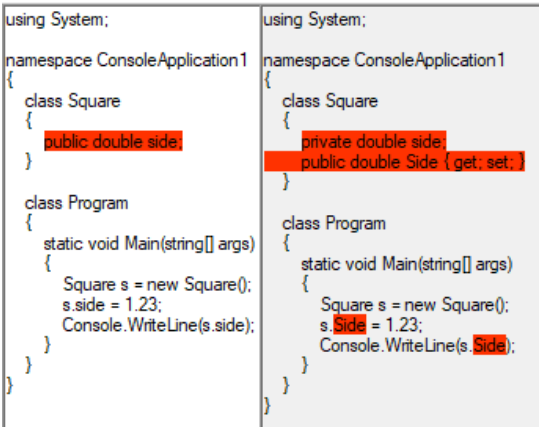


Fig. 8. Encapsulating field.

The refactoring is divided into 2 patterns: moving a method into the class and correcting names. Firstly, the number of calls is calculated, method is transferred if necessary, and the new location is highlighted. Secondly, if the current position of a method does not match the class where it is implemented, then the class name is replaced or added before the method call. The secondary pattern is referred using an instruction and the modifications performed by it are not highlighted.

As a result of the refactoring, we have achieved an improvement in response for a class (RFC) metric, but lack of cohesion in methods (LCOM) has also increased (Fig. 11).

5.2 Testing performance

A test bench has the following parameters: OS Windows 10 Pro, 64-bit system, Intel Core i5-6500 3.20GHz CPU, 16 GB RAM. Fig. 12 shows the chosen refactoring pattern, which have been used to test performance of the research prototype. We have applied that pattern to a code of Metrics Observer [25]. The project consists of 15 files written in C#.

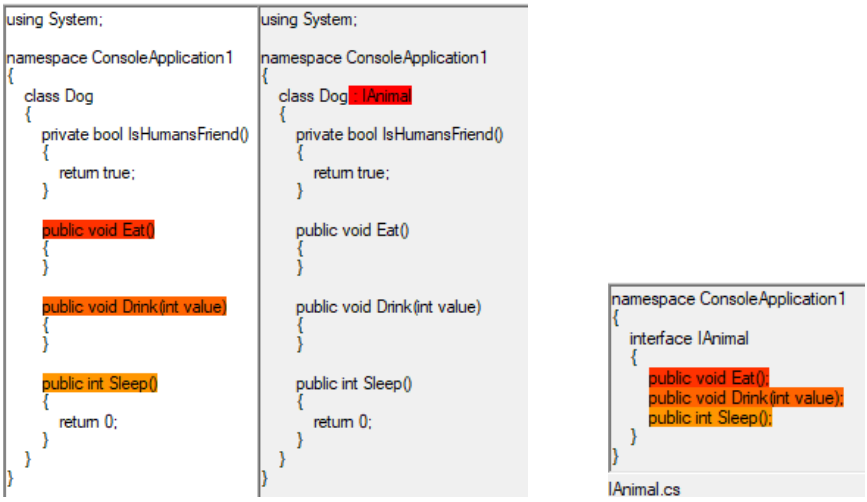


Fig. 9. Extracting interface and creating new file: a) source and refactored code; b) content of created file.

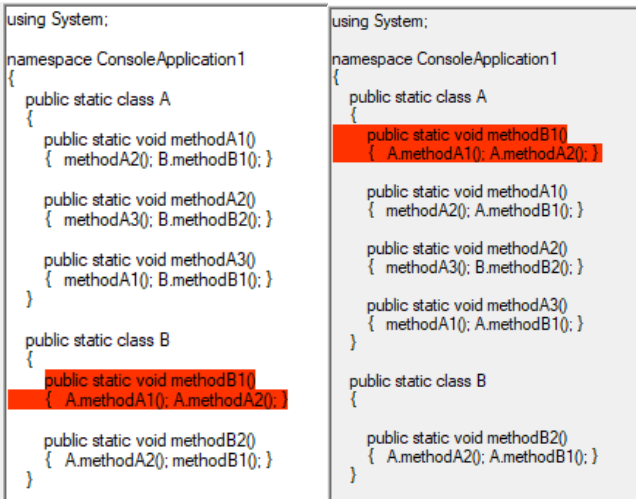


Fig. 10. Moving method and updating names: methodB1.

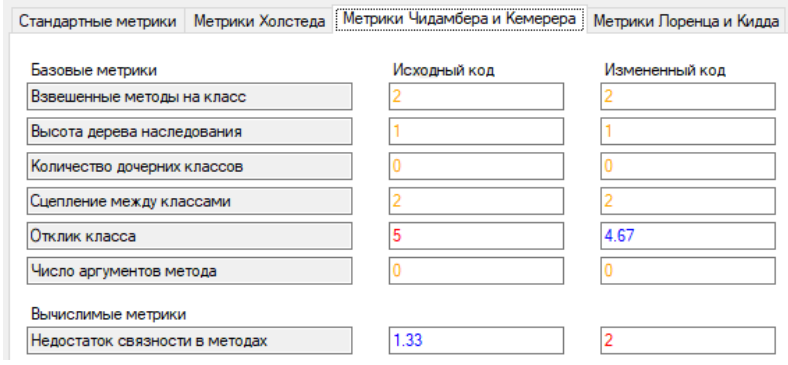


Fig. 11. Metrics comparison for methodB1.

```
<?xml version="1.0"?>
<ArrayOfRefactoring xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/
    <Refactoring xsi:type="Pattern">
        <Name>Вынести константы</Name>
        <Variables>
            <Variable xsi:type="Positions">constant</Variable>
            <Variable xsi:type="VariablesUnique">constantValue</Variable>
            <Variable xsi:type="Position">constantPut</Variable>
            <Variable>constantName<Value>tempConst</Value></Variable>
            <Variable>constantNumber</Variable>
            <Variable>langName</Variable>
        </Variables>
        <Search>
            <!--Определить язык-->
            <Entity xsi:type="Reserved">
            <!--1 В классе-->
            <Entity xsi:type="SkipTo">classSyKw</Entity>
            <!--Куда будет помещено объявление константы-->
            <Entity xsi:type="SkipTo">mainStructureSy</Entity>
            <Entity xsi:type="FindPosition">
                <Link>constantPut</Link>
                <Of>
            </Entity>
            <!--2 из блоков кода-->
            <Entity xsi:type="SkipTo">
            <!--если не конец класса-->
            <Entity xsi:type="If">
                <Check xsi:type="CheckExpression">
                    <Verifiable xsi:type="Not">
                </Check>
                <Description>
                    <!--3 найти константу-->
                    <Entity xsi:type="While">
                        <Check xsi:type="CheckExpression">
                        <Description>
                            <Entity xsi:type="SkipTo">
                            <!--4 если нашлась константа - запомнить значение-->
                            <Entity xsi:type="If">
                                <Check xsi:type="CheckType">const</Check>
                                <Description>
                                    <Entity xsi:type="SaveToken">constantValue</Entity>
                                    <Entity xsi:type="FindPosition">
                                        <Link>constant</Link>
                                        <Of>
                                            <Entity xsi:type="AcceptType">const</Entity>
                                        </Of>
                                    </Entity>
                                </Description>
                            </Entity>
                        </Description>
                    </Entity>
                </Description>
            </Entity>
            <!--5 если следующий класс или конец файла - завершение-->
        </Search>
        <Replace>
            <Entity xsi:type="InsertAfter">
            <Entity xsi:type="While">
                <Check xsi:type="CheckValue">constantValue</Check> <!--Пока в стеке есть значения-->
                <Description>
                    <!--Добавление объявления-->
                    <Entity xsi:type="InsertAfter">
                        <Position>constantPut</Position>
                        <Description>
                            <!--Получение номера значения константы с именем constantValue-->
                            <Entity xsi:type="VariablesNumber">constantValue<Link>constantNumber</Link></Entity>
                            <Entity xsi:type="If">
                        </Description>
                    </Entity>
                    <!--Замена константы на заданное имя-->
                    <Entity xsi:type="Insert">
                        <Position>constant</Position>
                        <Description>
                            <Entity xsi:type="NameNoSpace">constantName</Entity>
                            <Entity xsi:type="NameNoSpace">constantNumber</Entity>
                        </Description>
                    </Entity>
                    <Entity xsi:type="VariablesNext">constant</Entity>
                </Description>
            </Entity>
        </Replace>
    </Refactoring>

    <Refactoring xsi:type="Fragment">
        <Name>constDef</Name>
        <Variables>
        <Description>
    </Refactoring>
</ArrayOfRefactoring>
```

Fig. 12. Refactoring pattern fragment for transforming literals into constants.

Project files were automatically sorted in lexicographic order of their names and enumerated. Code fragments matching the refactoring pattern were found in 10 files. We measured the time taken by the main steps of the refactoring loop and number of logical lines of code for each file (Fig. 13).

As a result, the most time spent was equal to 589 ms. It was required to apply refactoring pattern to syntactic analyzer class (file 7) and calculate metric values. It took 213 ms to find code fragments in the file containing 4023 logical lines of code, and 337 ms to calculate and compare metric values. Despite the size of code, it had only 14 literals to be transformed into constants. However, the most amount of time spent for code modification step was required to change a code of token dictionary (file 2), which contained the largest number of unique literals: 102.

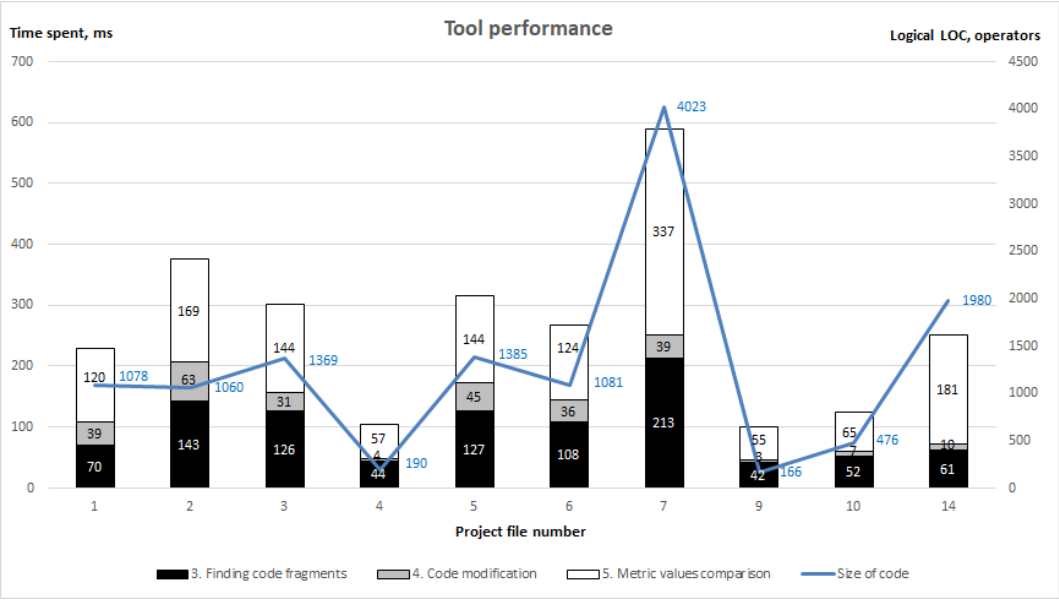


Fig. 13. Tool performance.

6. Discussion

Our tool requires language specifications in the DSL developed by a programmer. Nevertheless, it provides a generic tool for object-oriented languages and refactoring operations implementation. It may allow reducing time to deal with languages based on similar syntax and also create custom refactoring methods, make decisions regarding code modification and accumulate experience.

The developed design of a code refactoring tool that uses metric calculation and the created research prototype of our application demonstrated the possibility of:

- describing refactoring patterns using the developed DSL;
- generalized work with various object-oriented languages;
- comparing metrics when performing refactoring.

Our tool allows a developer to calculate metrics and refactor code using lexis and syntax descriptions in the DSL. Furthermore, it may be used for various versions, extensions, and new programming languages based on object-oriented paradigm.

However, despite the opportunity to deal with diverse languages and define a single refactoring pattern in several ways, it has not already proven that an arbitrary pattern could be described with the DSL. Another limitation is the complexity of working with text DSL:

- significant increase in the number of physical code lines compared to logical ones, including xml tags;
- reduced clarity due to high nesting and large amount of text;
- requirement of the DSL specification due to a large number of keywords;
- unavailability of an environment for writing code in the DSL.

Currently, the smallest pattern (Fig. 6) consists of 78 lines and 155 words, whereas the most complex one (Fig. 9) contains 434 lines and 655 words.

The performance depends on logical lines of code; complexity of code, pattern and language descriptions used; size of entities used in refactoring pattern such as set or dictionary.

7. Conclusion

Our application requires language specifications in the DSL developed by a programmer. Nevertheless, it provides a generic tool for object-oriented languages and refactoring operations implementation. It allows reducing time to deal with languages based on similar syntax and also create custom refactoring methods, make decisions regarding code modification and accumulate experience.

In this paper we have described an approach to implement code refactoring using its metrics calculation and refactoring patterns. The DSL, research prototype, 3 OOP languages specifications, and 6 typical refactoring patterns have been created. These examples have demonstrated the capabilities of this language.

Our tool may be used in organizations to refactor code and unambiguously evaluate its properties. Moreover, it may also be applied in educational institutions to verify and correct code written by students.

This study has a several possible directions for further activities. Firstly, an equivalent visual DSL and a suitable development environment for it should be created. Secondly, studying of the required language features for describing arbitrary refactoring can be necessary. Finally, proposing recommendations for pattern application based on code metrics and confirmation profitability of code modification may also be promising.

References

[1]. Качанов В.В., Ермаков М.К., Панкратенко Г.А., Спиридонов А.В., Волков А.С., Марков С.И. Технический долг в жизненном цикле разработки ПО: запахи кода. Труды ИСП РАН, том 33, вып. 6, 2021 г., стр. 95-110. DOI: 10.15514/ISPRAS-2021-33(6)-7. / Kachanov V.V., Ermakov M.K., Pankratenko G.A., Spiridonov A.V., Volkov A.S., Markov S.I. Technical debt in the software development lifecycle: code smells. Trudy ISP RAN/Proc. ISP RAS, 2021, vol. 33, issue 6, pp. 95-110 (in Russian). DOI: 10.15514/ISPRAS–2021–33(6)–7.

[2]. Sharma T., Efstathiou V., Louridas P., Spinellis D. Code smell detection by deep direct-learning and transfer-learning. Journal of Systems and Software, vol. 176, article no. 110936, 2021, pp.1-25. DOI: 10.1016/j.jss.2021.110936.

[3]. Сыромятников С. В., Бронштейн И. Е., Луговской Н. Л. Рефакторинг в рамках программного проекта. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 395-402. DOI: 10.15514/ISPRAS-2014-26(1)-16. / Syromyatnikov S. V., Bronshteyn I. E., Lugovskoy N. L. Refactoring on the Whole Project. Trudy ISP RAN/Proc. ISP RAS, 2014, vol. 26, issue 1, pp. 395-402 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-16.

[4]. Ivers J., Nord R. L., Ozkaya I., Seifried C., Timperley C. S., Kessentini M. Industry's cry for tools that support large-scale refactoring. In Proc. of the 44th International Conference on Software Engineering: Software Engineering in Practice, 2022, pp. 163-164. DOI: 10.1145/3510457.3513074.

[5]. Almogahed A., Mahdin H., Omar M., Zakaria N. H., Alawadhi A., Barraood S. O. Empirical Investigation of the Diverse Refactoring Effects on Software Quality: The Role of Refactoring Tools and Software Size. In Proc. of the 2023 3rd International Conference on Emerging Smart Technologies and Applications, 2023, pp. 1-6. DOI: 10.1109/eSmarTA59349.2023.10293407.

[6]. Golubev Y., Kurbatova Z., AlOmar E. A., Bryksin T., Mkaouer M. W. (2021) One Thousand and One Stories: A Large-Scale Survey of Software Refactoring (online). Available at: https://doi.org/10.48550/arXiv.2107.07357, accessed 05.05.2025.

[7]. Peruma A., AlOmar E. A., Newman C. D., Mkaouer M. W., Ouni A. Refactoring Debt: Myth or Reality? An Exploratory Study on the Relationship Between Technical Debt and Refactoring. In Proc. of the 2022 IEEE/ACM 19th International Conference on Mining Software Repositories, 2022, pp. 127-131. DOI: 10.1145/3524842.3528527

[8]. Li Z., Avgeriou P., Liang P. A Systematic Mapping Study on Technical Debt and its Management. Journal of Systems and Software, vol. 101, 2015, pp. 193-220. DOI: 10.1016/j.jss.2014.12.027.

[9]. Panigrahi R., Kuanar S. K., Kumar L. Application of Naïve Bayes classifiers for refactoring Prediction at the method level. In Proc. of the 2020 International Conference on Computer Science, Engineering and Applications, 2020, pp. 1-6. DOI: 10.1109/ICCSEA49143.2020.9132849.

[10]. Lambiase S., Cupito A., Pecorelli F., De Lucia A., Palomba F. Just-In-Time Test Smell Detection and Refactoring: The DARTS Project. In Proc. of the 2020 IEEE/ACM 28th International Conference on Program Comprehension, 2020, pp. 441-445. DOI: 10.1145/3387904.3389296.

[11]. Perera J., Tempero E., Tu Y.-C., Blincoe K. Quantifying Requirements Technical Debt: A Systematic Mapping Study and a Conceptual Model. In Proc. of the 2023 IEEE 31st International Requirements Engineering Conference, 2023, pp. 123-133. DOI: 10.1109/RE57278.2023.00021.

[12]. Tavares C., Ferreira F., Figueiredo E. A Systematic Mapping of Literature on Software Refactoring Tools. In Proc. of the XIV Brazilian Symposium on Information Systems, 2018, article no. 11, pp. 1-8. DOI: 10.1145/3229345.3229357.

[13]. Murphy-Hill E., Black A. P. Refactoring Tools: Fitness for Purpose. IEEE Software, 2008, vol. 25, issue 5, pp. 38-44. DOI: 10.1109/MS.2008.123.

[14]. Zhang Z., Xing Z., Xu X., Zhu L. RIdiom: Automatically Refactoring Non-Idiomatic Python Code with Pythonic Idioms. In Proc. of the 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings, 2023, pp. 102-106. DOI: 10.1109/ICSE-Companion58688.2023.00034.

[15]. Zhang Y., Li C., Shao S. ReSwitcher: Automatically Refactoring Java Programs for Switch Expression. In Proc. of the 2021 IEEE International Symposium on Software Reliability Engineering Workshops, 2021, pp. 399-400. DOI: 10.1109/ISSREW53611.2021.00108.

[16]. Iannone E., Pecorelli F., Di Nucci D., Palomba F., De Lucia A. Refactoring Android-specific Energy Smells: A Plugin for Android Studio. In Proc. of the 2020 IEEE/ACM 28th International Conference on Program Comprehension, 2020, pp. 451-455. DOI: 10.1145/3387904.3389298.

[17]. Корзников А. О., Дацун Н. Н. Методы и средства расчета и применения метрик кода программных продуктов: систематическое картографирование литературы. Известия СПбГЭТУ «ЛЭТИ», том 17, вып. 8, 2024 г., стр. 48-64. DOI: 10.32603/2071-8985-2024-17-8-48-64. / Korznikov A. O., Datsun N. N. Methods for Calculation and Application of Software Code Metrics: A Systematic Mapping Study. LETI Transactions on Electrical Engineering & Computer Science, 2024, vol. 17, issue 8, pp. 48-64 (in Russian). DOI: 10.32603/2071-8985-2024-17-8-25-64.

[18]. Colakoglu F. N., Yazici A., Mishra A. Software Product Quality Metrics: A Systematic Mapping Study. IEEE Access, vol. 9, 2021, pp. 44647-44670. DOI: 10.1109/ACCESS.2021.3054730.

[19]. Mshelia Y. U., Apeh S. T., Edoghogho O. A comparative assessment of software metrics tools. In Proc. of the 2017 International Conference on Computing Networking and Informatics, 2017. P. 1-9. DOI: 10.1109/ICCNI.2017.8123809.

[20]. Agnihotri M., Chug, A. A Systematic Literature Survey of Software Metrics, Code Smells and Refactoring Techniques. Journal of Information Processing Systems, 16(4), 2020, pp. 915-934. DOI: 10.3745/JIPS.04.0184.

[21]. Fernandes S., Aguiar A., Restivo A. LiveRef: a Tool for Live Refactoring Java Code. In Proc. of the 37th IEEE/ACM International Conference on Automated Software Engineering, 2022, article no. 161, pp. 1-4. DOI: 10.1145/3551349.3559532.

[22]. Mooij A. J., Ketema J., Klusener S., Schuts M. Reducing Code Complexity through Code Refactoring and Model-Based Rejuvenation. In Proc. of the 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering, 2020, pp. 617-621. DOI: 10.1109/SANER48275.2020.9054823.

[23]. Luciv D. V., Koznov D. V., Shelikhovskii A. A., Romanovsky K. Yu., Chernishev G. A., Terekhov A. N., Grigoriev D. A., Smirnova A. N., Borovkov D. V., Vasenina A. I. Interactive Near Duplicate Search in Software Documentation. Programming and Computer Software, 2019, vol. 45, pp. 346-355. DOI: 10.1134/S0361768819060045.

[24]. Dallal J. Al, Abdulsalam H., AlMarzouq M., Selamat A. Machine Learning-Based Exploration of the Impact of Move Method Refactoring on Object-Oriented Software Quality Attributes. Arabian Journal for Science and Engineering, 2024, vol. 49, pp. 3867-3885. DOI: 10.1007/s13369-023-08174-0.

[25]. Корзников А. О., Дацун Н. Н. Разработка приложения для получения метрик программного продукта на языке объектно-ориентированного программирования. Вестник Пермского университета. Математика. Механика. Информатика, вып. 3 (62), 2023 г., стр. 76-84. DOI: 10.17072/1993-0550-2023-3-76-84. / Korznikov A.O., Datsun N.N. Program Realization for Code Metrics Calculation in Object-Oriented Programming Language. Bulletin of Perm University. Mathematics. Mechanics. Computer Science, 2023, issue 3(62), pp. 76-84. (in Russian). DOI: 10.17072/1993-0550-2023-3-76-84.

Информация об авторах / Information about authors

Артем Олегович КОРЗНИКОВ – магистрант физико-математического института Пермского государственного национального исследовательского университета, бакалавр прикладной математики и информатики. Сфера научных интересов: метрики кода, объектно-ориентированные языки программирования, предметно-ориентированные языки, рефакторинг кода.

Artem Olegovich KORZNIKOV – BSc (Applied Mathematics and Computer Science), Master's student of the Department of Physics and Mathematics of PSU. Research interests: code metrics, object-oriented programming languages, domain specific languages, code refactoring.

Наталья Николаевна ДАЦУН является приглашенным преподавателем кафедры информационных технологий в бизнесе Национального исследовательского университета «Высшая школа экономики», Пермь; доцент кафедры прикладной информатики, информационных систем и технологий Пермского государственного гуманитарно-педагогического университета, кандидат физико-математических наук, доцент. Ее научные интересы включают метрики кода, объектно-ориентированный анализ и проектирование.

Natalya Nikolaevna DATSUN – Cand. Sci. (Phys.-Math,), visiting lecturer of Department of Information Technology in Business of the National Research University Higher School of Economics, Perm; Associate Professor, Department of Applied Informatics, Information Systems and Technologies, Perm State Humanitarian Pedagogical University. Her research interests include code metrics, object-oriented analysis, and design.

DOI: 10.15514/ISPRAS-2025-37(5)-12

Improving Image Analysis and Processing Performance on the RISC-V Platform with Lichee Pi 4A

N.I. Cherepanov, ORCID: 0009-0001-9135-9654 <cherepanov.ni@edu.spbstu.ru>
N.O. Stepina, ORCID: 0009-0001-4740-637X <gubenko_no@spbstu.ru>
I.V. Nikiforov, ORCID: 0000-0003-0198-1886 <nikiforov_iv@spbstu.ru>

Peter the Great St. Petersburg Polytechnic University,
29, Polytechnicheskaya st., St. Petersburg, 195251, Russia.

Abstract. The study explores optimization methods for improving image processing performance on the RISC-V platform with Lichee Pi 4A. The research focuses on real-time video processing within a microservice-based self-service system. Several existing optimization strategies are considered and evaluated, including neural network model optimization, hardware acceleration using RVV vector instructions and leveraging the built-in Neural Processing Unit (NPU). The profiling results on existing strategies indicate that object detection and feature extraction consume the most computation resources. In order to eliminate the performance gap, the model quantization to INT8 format is implemented, that allows to reduce memory usage and inference latency. Additionally, a modified ONNX Runtime version is deployed to support NPU acceleration. These improvements led to 75% reduction in model size and a 35% decrease in inference latency. The study concludes that hardware-aware optimizations significantly enchase performance on the RISC-V (Lichee Pi 4A) platform. The main issue encountered is the low processing speed on Lichee Pi 4A, with a current frame rate of only 0.05 FPS, which in unsuitable for practical usage.

Keywords: RISC-V; Lichee Pi 4A; image processing; neural network; vectorization; NPU; ONNX Runtime; performance optimization; real-time processing.

For citation: Cherepanov N. I., Stepina N. O., Nikiforov I. V. Improving image analysis and processing performance on the RISC-V platform with Lichee Pi 4A, Proceedings of the Institute for System Programming of the RAS, vol. 37, issue 5, 2025, pp. 157-172. DOI: 10.15514/ISPRAS-2025-37(5)-12.

Повышение производительности анализа и обработки изображений на платформе RISC-V с помощью Lichee Pi 4A

Н.И. Черепанов, ORCID: 0009-0001-9135-9654 <cherepanov.ni@edu.spbstu.ru>
Н.О. Степина, ORCID: 0009-0001-4740-637X <gubenko_no@spbstu.ru>
И.В. Никифоров, ORCID: 0000-0003-0198-1886 <nikiforov_iv@spbstu.ru>

Санкт-Петербургский политехнический университет Петра Великого,
195251, Россия, Санкт-Петербург, Политехническая улица, д. 29.

Аннотация. В исследовании изучаются методы оптимизации для повышения производительности обработки изображений на платформе RISC-V с использованием Lichee Pi 4A. Исследование сосредоточено на обработке видео в режиме реального времени для системы самообслуживания, которая реализована в виде микросервисного приложения. Рассматриваются и оцениваются стратегии оптимизации, включая оптимизацию модели нейронной сети, аппаратное ускорение с использованием векторных инструкций RVV и использование встроенного ускорителя для нейронных сетей (NPU). Результаты профилирования существующих стратегий показывают, что обнаружение объектов и извлечение признаков потребляют большую часть вычислительных ресурсов. Чтобы устранить разрыв в производительности, реализовано квантование модели в формат INT8, что позволяет сократить использование памяти и задержку вывода. Кроме того, развернута модифицированная версия ONNX Runtime для поддержки ускорения NPU. Эти улучшения привели к уменьшению размера модели на 75% и уменьшению задержки вывода на 35%. В исследовании делается вывод, что аппаратно-ориентированные оптимизации значительно повышают производительность на платформе RISC-V (Lichee Pi 4A). А также определена основная проблема практического применения разработанного решения на Lichee Pi 4A, связанная с низкой скоростью обработки данных: текущая частота кадров составляет всего 0,05 FPS.

Ключевые слова: RISC-V; Lichee Pi 4A; обработка изображений; нейронная сеть; векторизация; NPU; ONNX Runtime; оптимизация производительности; обработка в реальном времени.

Для цитирования: Черепанов Н. И., Степина Н. О., Никифоров И. В. Повышение производительности анализа и обработки изображений на платформе RISC-V с помощью Lichee Pi 4A, Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 157–172 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)–12.

1. Introduction

Modern and young open RISC-V [1] architecture is widely used in embedded systems and high-performance computing. However, when it comes to computer vision [2] and image processing, the platforms, that implements the RISC-V architecture, face several challenges. Well-established architectures such as x86 and ARM [3] are free of those challenges because of years of development and thousands of researchers and developers involved.

One of the main challenges of using RISC-V (especially on Lichee Pi 4A) for image and video processing is low framerate (FPS) when processing video streams, which is critical for object detection and classification.

For production lines and environments, where, for example, robotic arms are used, that are equipped with vision systems, video processing plays a crucial role in object recognition (Fig. 1). Computer vision relies heavily on video stream processing [4] as working with dynamic scenes requires real-time object recognition and rapid system response to changes. This is particularly important in fields such as retail, medical diagnostics and autonomous systems, where the accuracy and speed of frame analysis directly impact decision-making. Transitioning from standard processors to RISC-V platforms could significantly reduce manufacturing costs due to their open-source nature and hardware flexibility in comparison to traditional hardware and software design.

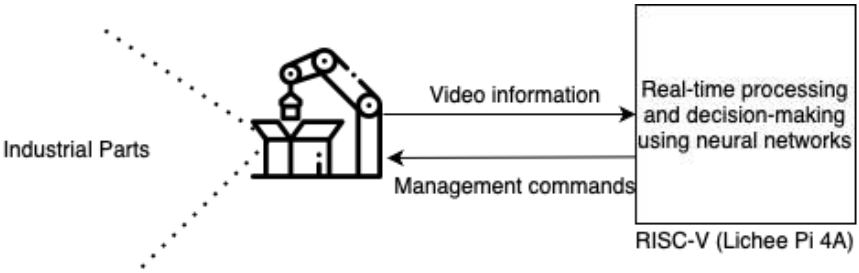


Fig. 1. Testing environment - computer vision system.

There are the following existing implementations of RISC-V on the market: Lichee Pi 4A, Mango pi MPI-MQ1, Milk-V Pioneer, Banana Pi BPI-K1, VisionFive 2, GiFive Unmatched. Each of these platforms varied in terms of performance, available features and suitability for machine vision applications (Table 1).

The Lichee Pi 4A board served as the hardware platform for this project, following a task proposed by an industrial partner. The goal of the work includes evaluation of the performance characteristics and evaluating if Lichee Pi 4A is suitable for practical applicability of this specific RISC-V implementation in real-time machine vision scenarios. Compares to other boards, Lichee Pi 4A offered a balanced combination of high CPU frequency, a powerful GPU and a dedicated NPU, making it suitable for neural inference tasks such feature extraction.

Table 1. Comparison of characteristics of single RISC-V Boards.

Model	CPU	CPU Freq.	GPU	NPU	RAM	Price
Mango Pi	Allwinner D1 (C906, RISC-V)	1.0GHz	-	-	1GB DDR3	~$20
Lichee Pi	T-Head TH1520 (4xC910)	2.0GHz	Imagination BXM-4-64	4 TOPS	up to 16GB LPDDR4X	~$119
Mikl-V	SOPHON SG2042 (64xC920)	up to 2Ghz	-	-	up to 128GB DDR4	~$1000
Banana Pi	SpacemiT K1 (6xX60)	-	IMG BXE-2-32	2 TOPS	up to16 GB LPDDR4	~$100
VisionFive 2	StarFive JH7110 (4xU74)	1.5GHz	IMG NXE-4-32	-	up to 8GB LPDDR3	~$70
HiFive	SiFive U740 (4xU74 +S7 core)	1.2GHz	-	-	16GB DDR4	~$665

As a result of the testing and evaluating the performance in the article it is concluded that Lichee Pi 4A lags in performance, especially in real-time processing. This is not due to RISC-V flaws in the architecture itself, but rather its relative novelty: high-performance chips are still in development and many essential software tools have not been ported yet.

As far as there is no direct access to industrial systems, article authors created a development environment for retail domain. There is a microservise application [5] developed, where video processing serves as the functionality. Based on this system, various optimization approaches are considered and evaluating. The system consists of three main microservices [6]:

- backend service - responsible for video stream processing, object detection and managing the consumers requests;

- frontend service - provides the user interface and displays the video stream;
- database service - stores product data, including names, prices and categories.

The main goal of testing system, that is used for performance evaluation, is to automatically identify the products taken by the customer and generate a shopping cart for checkout. However, its current implementation, the video processing speed is only 0.05 FPS, making the system unsuitable for practical use. To ensure successful, the processing speed must reach 30 FPS [7].

Thus, the key objective of this study is to increase the performance of the computer vision system on the Lichee Pi 4A platform to 30 FPS. To achieve this, the following steps are necessary [8]:

- optimizing the neural network model for object detection;
- improving frame processing while considering the capabilities of the RISC-V platform;
- utilizing hardware accelerators such as NPU, SIMD and RISC-V Vector Extensions (RVV) [9].

To evaluate the system's real-time performance, the frames per second (FPS) metric is measured using Python high-resolution timer. The procedure includes the following steps:

- at the beginning of each frame-processing cycle, the start timestamp is recorder;
- the frame undergoes all stags of processing, including acquisition, processing, neural network inference and postprocessing;
- upon completion, the end timestamp is recorded;
- the time taken for a single frame is computed as the difference between the end and start time;
- instantaneous FPS is calculated as the reciprocal of the frame time;
- this process is repeated for a large number of frames and the average FPS is derived by averaging the collected values.

To assess the computational load of operations, CPU usage is analyzed at each stage of processing. The 15-20% allocation for preprocessing is determined by comparing the total processing time with the time spent specifically on this stage across several experiments.

In order to understand hardware and software design (co-design) of the experiment stand, that is critical for performance evaluation, let's consider every part separately.

2. Research

Modern research it the field of image processing on the RISC-V platform demonstrates a growing interest in optimizing performance and energy efficiency, especially for embedded systems and devices with limited resources. This chapter examines the key work on this topic, as well as highlights their main achievements and limitations.

In [10], a hardware accelerator for YOLOv3-Tiny using RISC-V SoC was proposed. The authors achieve a bandwidth of 21.6 GOPS/s, but note limitations associated with frequent memory access.

The article [11] compares various models (SOLO, SSD, Faster RUN) on the SiFive U540 platform. YOLOv3 and SSD-MobileNet showed the best results, which confirms the importance of choosing a model for a specific hardware platform.

The work [12] demonstrates the advantages of vector instructions to speed up CNN operations. The authors note that increasing the length of the vector (VLEN) does not always lead to a proportional increase in performance due to memory limitations.

In [13], the use of TVM for quantized RISC-V models with the P extension is investigated. The results show an acceleration of 2.7 – 7.0 times compared to FP32, which highlights the potential of quantization for RISC-V.

3. Platform's hardware equipment

The project is implemented using the Lichee Pi 4A - a single-board computer based on the T-Head TH1520 processor. Its key specifications include:

- processor – 4-core RISC-V C920 (up to 1.85 GHz) with SIMD and RVV 0.7.1 support;
- graphics – 50 GFLOPS Imagination BXM-4-64 GPU (supports OpenGL ES 3.x and Vulcan);
- NPU – 4 TOPS performance for accelerating AI computations;
- RAM – up to 16 GB LPDDR4/4x.

The T-Head TH1520 processor, developed by Alibaba Group's semiconductor division, is designed for embedded systems with high computational demands. It features an optimized L1 and L2 cache hierarchy, which plays a crucial role in processor performance. The L1 cache is split into separate instruction and data caches, allowing for faster access to frequently used data and reducing latency. The L2 cache, being larger and shared among cores, helps mitigate memory bottlenecks by storing recently accessed data, reducing the need for frequent main memory accesses. This cache structure significantly improves processing speed, particularly in image analysis and video processing tasks, where rapid data retrieval is essential. The BXM-4-64 GPU provides hardware-accelerated rendering and supports 4K displays. However, for machine learning tasks and other algorithms that require massive parallel computing, it is recommended to use NPU, since its performance higher than the GPU capabilities in similar workloads.

4. Software architecture

ONNX Runtime is a high-performance inference engine designed to execute machine learning models in the ONNX (Open Neural Network Exchange) format [14]. It provides hardware acceleration and optimization techniques, making it suitable for deployment across various platforms, including CPU, GPU and specialized accelerators.

The project uses ONNX Runtime for model execution because TensorFlow, PyTorch and other major ML libraries are not officially ported to RISC-V. TensorFlow Lite for Microcontrollers has been ported to RISC-V architecture, but this is just a lightweight version. Porting the full version of TensorFlow to RISC-V requires the use of cross-compilers and additional settings, which is confirmed by the documentation of the RISE project. PyTorch also has no official support for the RISC-V architecture. There are initiatives to port PyTorch to RISC-V, such as the pythorch-riscv64 project, which provides pre-built packages for RISC-V. However, these solutions are experimental and are not part of the official PyTorch release. In addition, discussions on the PyTorch forums confirm that official support for RISC-V is in plans but has not yet been implemented. Since there is no built-in support for these platforms in RISC-V, ONNX provides a universal solution that allows you to export models trained in various environments (for example, PyTorch or TensorFlow) to ONNX format and then efficiently execute them on RISC-V hardware.

Key reasons for choosing ONNX on Lichee Pi 4A are listed below.

1. Cross-platform compatibility – ONNX models can be exported from multiple ML frameworks.
2. Hardware acceleration – ONNX Runtime optimizes inference through quantization, graph optimizations and hardware-specific execution provides.
3. Lack for TensorFlow/PyTorch support – since these frameworks are not available on RISC-V, ONNX is the best alternative.
4. Support for custom execution providers – while ONNX Runtime does not native support TH1520 NPU, it allows experimentation with custom providers like ShlExecutionProvider for potential acceleration.

YOLOv8n (You Only Look Once, version 8, nano model) is a deep learning model designed for real-time object detection [15]. It balances accuracy and speed, making it suitable for embedded systems like the Lichee Pi 4A. The model is exported in ONNX format for compatibility with ONNX Runtime.

Key features of YOLOv8n:

- single-stage detection – the model predicts object location and classifications in a single pass, ensuring fast inference;
- optimized for edge devices – the "small" version is designed for efficiency, making it suitable for resource – limit platforms;
- flexibility – it can be quantized to INT8 for acceleration on NPU, though additional steps are needed for TH1520 support.

YOLOv8n followed a CSP-based architecture [16] and included three main components:

- backbone (C2f + CBS) – extracted features at multiple scales using convolutional layers with residual connections;
- neck (PAN-FPN) – aggregated multi-scale feature maps using anchor-free detection;
- head – prediction object classes and bounding boxes directly from feature maps using anchor-free detection.

This lightweight design allowed the model to maintain good detection accuracy with reduced latency and memory usage.

To train the object detection model, a custom dataset is created. The dataset consists of N products categories, each containing 500 images, a total of 2800 images are used for training and validation of the model, approximately 85MB on disk, collected from various online sources [17]. The dataset is prepared in the YOLO format, which includes:

- images – the raw images containing objects of interest;
- annotation files – each image has a corresponding text file with bounding box coordinates and class labels in YOLO format.

The annotation process involved:

- collecting images – downloading diverse product images to cover different angles, lighting conditions, and backgrounds;
- manually labeling objects – using LabalImg and other annotation tools to draw bounding boxes around objects and assign category labels.

The dataset images vary in resolution. All images are stored in 24-bit RGB color format with a DPI of 72. This dataset is used to train YOLOv8n, optimizing it for real-world object detection in the system.

As the metrics below show, this amount of data is enough to detect objects, but for more important tasks, for example in the field to medicine, where accuracy should be close to 1, an order magnitude more images are needed [18].

After training, the model achieved high accuracy. The average reached 0.993, indicating an almost perfect match between predicated and actual objects.

On the Precision-Recall Curve [19] (Fig. 2), the curve for most classes stayed close to the upper-right corner, confirming high precision along with excellent recall.

On the Recall-Confidence Curve [20] (Fig. 3), all classes maintained high recall up to a confidence threshold of 0.85-0.9, meaning the model detected almost all object even at high confidence levels.

The system follows a structured pipeline for image processing and object detection, that is described step by step below.

1. Preprocessing – normalization, resizing, and noise reduction.
2. Embedding extraction – converting the image into a vector representation.
3. Inference – running the neural network for detection and classification.
4. Postprocessing – interpreting and visualizing the results.

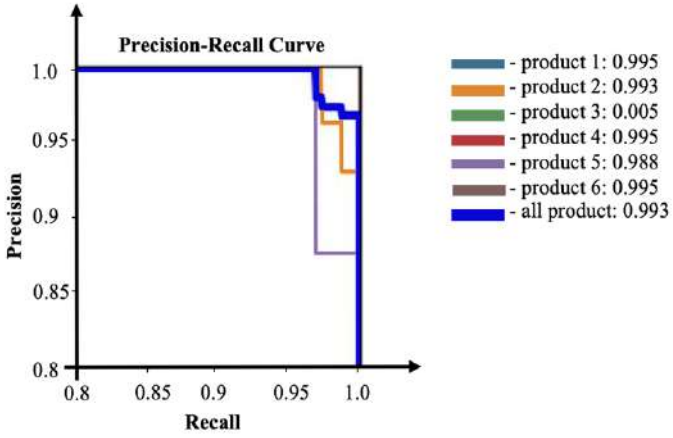


Fig. 2. Precision-Recall Curve for testing dataset of object recognition.

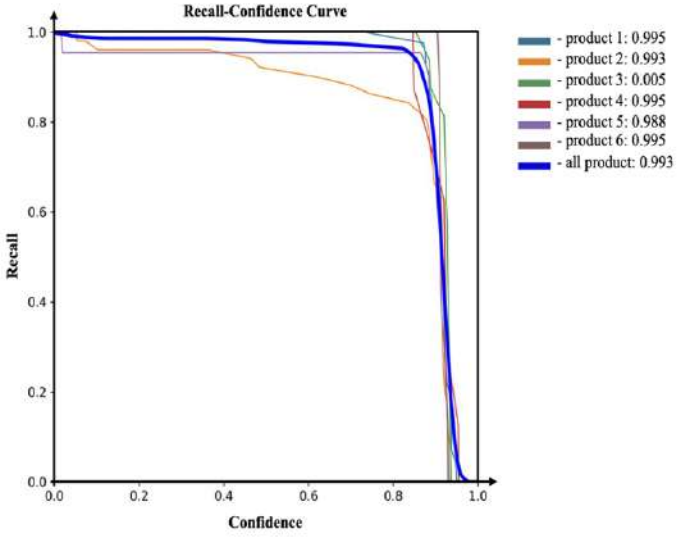


Fig. 3. Recall-Confidence Curve for testing dataset of object recognition.

5. Bottleneck analysis

Let's consider video processing steps and how they are implemented.

The OpenCV library is utilized for video stream capture and preprocessing [21], providing user-friendly interfaces for handling video sources and image processing. Object detection is performed using the ONNX version of YOLO [22], executed via ONNX Runtime.

Video acquisition is handled using OpenCV through the cv2.VideoCapture object. The resolution parameters for the video stream are defined this loop as listing 1:

Listing 1. Loop video stream capture

```
cap = cv2.VideoCapture(1)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
```

This configuration allows capturing frames at a resolution of 640x480 pixels in real time. The value 1 in VideoCapture(1) specifies that an external camera is being used. However, the resolution of 640x480 indicated in the article formally falls under the category of "low" according to GOST 51558-2014, where the threshold is considered to be a resolution of up to 756x576 pixels. In addition, the choice of this resolution in the article is not due to an attempt to achieve an industrial level of quality, but to the desire to demonstration the operability of the entire system at a prototype level with low hardware capabilities. For industrial implementation, the solution can be adapted to a higher camera resolution that meets requirements of GOST with more efficient hardware at the same time. With our current experiment we see, that even for low picture resolution the recognition speed is not enough for industrial tasks.

Object detection is performed using YOLO model in ONNX format. The ONNX Runtime library in used for inference and preprocessing steps include below items.

1. Conversion to a Pillow object.
2. Resizing to 640x640 pixels (matching the YOLO model input format).
3. Pixel value normalization and array reshaping.

A typical video processing pipeline consists of the following key stages [23] listed below.

1. Video capture (from a camera or disk).
2. Preprocessing (video pipeline).
3. Object detection (detector).
4. Main data processing (post processing).
5. Database search (search).

Analyzing the workload at each stage helps identify the most resource-intensive operations and determine bottlenecks that affect system performance. The percentage values in the table are obtained through profiling and benchmarking of each processing stage. These are relative values from the total time. The time share is measured by running the video processing pipeline on the Lichee Pi 4A and recording the execution time for each step. Profiling and logging tools are used to analyze performance bottlenecks, and running tests multiple times ensures consistency of the results shown in Table 2.

Table 2. Time distribution and main limitations for video processing stages.

Processing stage	Time share	Main limitations
Video capture	0.10%	Depends on camera I/O speed
Preprocessing	36.27%	Resize, normalization CPU-bound
Object detection	61.24%	Low CPU processing speed
Main data processing	1.80%	Decode CPU-bound
Database search	0.04%	Scale with database size

To identify bottleneck in system performance, a detailed analysis is conducted using the gprof tool. The primary focus is on the following aspects:

- function execution time – measuring the time spent on key image processing stages;
- CPU and NPU workload distribution – analyzing hardware accelerator utilization;
- cache efficiency – evaluating the impact of caching on data processing speed.

Profiling is performed on real video streams with a resolution of 640x480 pixels. The backend service used for testing is ran on the Lichee Pi 4A.

5.1. Video capture and pre-processing

The video capture and pre-processing stage involves reading and decoding the video stream [24]. The main workload comes from continuous data writing and reading, which can quickly fill the cache memory. A limited cache size may cause additional delays due to frequent access to RAM.

5.2. Object detection

Detection is the first stage where neural network algorithms are applied [25]. It runs faster on a GPU, but if GPU acceleration is unavailable, the CPU must handle the workload, creating significant pressure on processor cores. On the Lichee Pi 4A, the built-in IPU can be used, but integrating it with ONNX Runtime presents certain challenges:

- manual model conversion and low-level integration are required;
- hardware support is limited;
- the lack of documentation and stable tools (HHB, SDK) complicates debugging.

5.3. Inference

This stage involves running deep neural network models [26]. The main workload is typically handled by the GPU, but on the RISC-V platform with RVV, some vector processing operations can be offloaded to hardware, reducing dependence on the GPU. However, the lack of stable OpenCL/Vulkan drivers for the GPU remains an issue.

5.4. Vectorization and embedding

After inference, feature extraction is required. This stage demands intensive computation. The use of SIMD and RVV vector instructions could speed up the process [27], but current implementations do not always take full advantage of these capabilities.

5.5. Database search

This stage places a load on memory and storage. If the database is stored on a device with limited memory, frequent disk access can cause delays. However, in the current project, this is not a critical issue that needs immediate resolution.

Future following improvements are possible:

- implementation of multi-level caching;
- use of memory-mapped files;
- optimization of data structures for RISC-V architecture;
- vector quantization to reduce memory footprint.

5.5. Diagnosing problems

Video processing challenges on the Lichee Pi 4A stem from both hardware limitations and software inefficiencies. Optimizations such as SIMD/RVV utilization, hardware NPU acceleration, and advanced memory management algorithms can significantly improve system performance.

Diagnostics can be performed at different levels:

- hardware level – the Lichee Pi 4A is based on a relatively new architecture and its processor implementation differs from more established platforms. This leads to potential inefficiencies due to immature compiler optimizations, incomplete hardware support, and

limited documentation;

- software level – many libraries and frameworks have not yet been fully ported to RISC-V, leading to compatibility issues and suboptimal performance. Additionally, the software stack itself can often be optimized further, reducing redundant computations and improving overall efficiency.

6. Optimization methods

Optimizing image processing is an important aspect that helps reduce computation time [28], lower CPU load and improve overall system performance. This section discusses various optimization techniques, from using hardware instructions to implementing multithreaded data processing.

6.1. Using RVV for preprocessing and preprocessing

The RVV vector instruction extension enables parallel computing (Fig. 4), which is particularly useful for matrix and tensor operations is neural network models [29] This method significantly speeds up tasks involving large amounts of data, such as image transformation, normalization, and convolution.

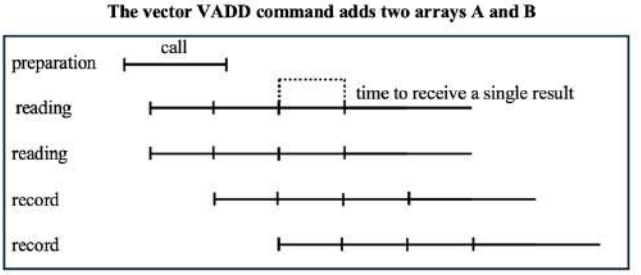


Fig. 4. Time diagram of vector addition of two arrays.

Applying RVV (RISC-V Vector Extension) can notably accelerate both data preprocessing (such as normalization, filtering and image transformations) and the inference stage of deep learning models. For example, operations like matrix multiplication within convolutional layers benefit greatly from vectorized execution [30]. Unlike scalar processing (Fig. 5), which handles data elements one at a time, vector registers in RVV enable simultaneous execution of multiple operations within a single CPU cycle. It allows you to load a bunch of values into a vector register and simultaneously perform operations on them.

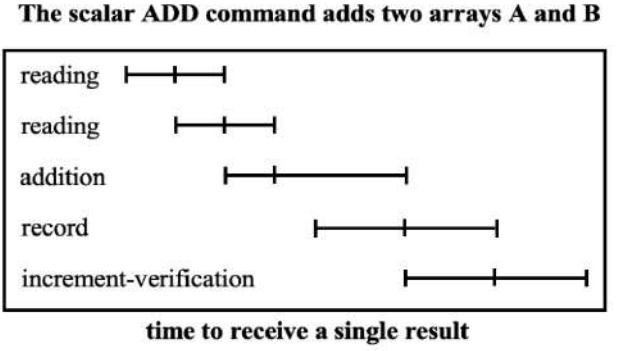


Fig. 5. Time diagram of vector addition of two arrays.

Since the analysis of the video processing pipeline from section IV showed that it is necessary to speed up the work of not only the inference model, but also postprocessing and preprocessing, since they also significantly load the system. To do this, we use vectorization of calculations. On Fig. 6

the color of rectangle represents the load factor of the module. Green color – is low level of load, yellow color – is medium load and red color – is the highest loaded modules, that requires a lot of hardware resources.

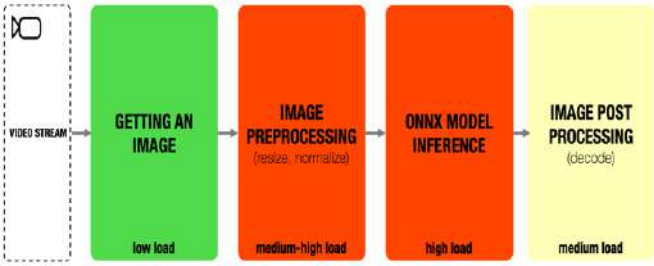


Fig. 6. Video processing pipeline.

To apply RVV in an experimental application, functions for preprocessing and postprocessing are written in C using an intrinsic. An important clarification is that the Lichee Pi 4A has the RVV 0.7.1 standard, which does not have auto-vectoring. Therefore, the RVV code is cross compiled into an executable file and functions are inserted into Python code using the Cpython library.

At the preprocessing stage (Fig. 7), image scaling, normalization and formatting operations are performed to feed into the neural network model. These steps include:

- resizing the image (cv2.resize);
- normalization of pixel values to the range [0.0, 0.1];
- channel rearrangement (CHW).

Using RVV allows to vectorize channel normalization and transformation operations. Instead of sequentially processing each pixel, RVV loads a vector of pixels and applies division and transpose operations in parallel.

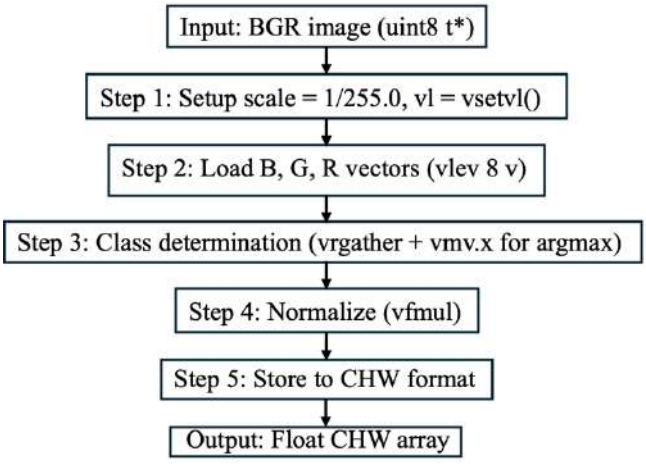


Fig. 7. Block diagram of the preprocessing function.

Postprocessing includes (Fig. 8) processing the output tensor of the model: threshold filtering, coordinate recalculation and preparation on the final list of objects. RVV is used to vectorize the following operations:

- extracting and converting coordinates of boxes (x1, x2y, x2, y2);
- calculation of confidence (np.max);
- finding the class (np.argmax);

- filtering by threshold;
- converting coordinates to pixels.

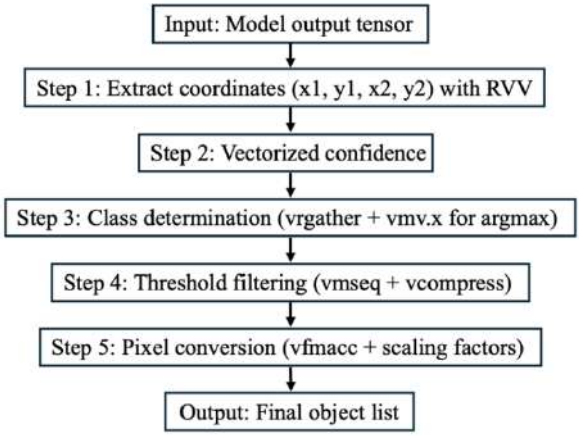


Fig. 8. Block diagram of the postprocessing function.

The following RVV intrinsics were used:

- vsetvl_e32m4 – sets the length of the vector for operations with 32-bit elements using 4 registers (m4). Automatically determining the maximum possible length for the remaining data.
- vsle_v_u8m1 – page loading of 8-bit unsigned integers in 3-byre increments. Allows you to load color components from an alternating format.
- vwaddu_vx_u16m2/vwaddu_vx_u32m4 – is an unsigned bit depth extension with a zero extension. What you need to convert to float.
- vfcvt_f_xu_v_f_32m4 – conversion of unsigned 32-bit integers to float32. It is necessary to maintain accuracy during normalization.
- vfmul_vf_32m4 – vector multiplication for normalization. Multiplies each element of the vector by 1/255 in one operation.
- vse_v_f32m4 – batch saving of 32-bit float values. This is necessary for the correct location of the data in the CHW format.

The postprocessing function has been optimized in a similar way.

By applying RVV instruction image processing, it was possible to achieve some speed improvements (Table 3). Thus, RVV becomes an excellent optimization tool both for processing the data preparation stages before launching the neural network and for subsequent processing of the results. Moreover, compared OpenCV vector methods the speed increases by about 2 times after using RVV. And by an order of magnitude compared to scalar methods.

Table 3. RVV application results.

	Time using OpenCV scalar functions (sec)	**Time using OpenCV vector functions (sec)**	**Out optimization option (sec)**
Preprocessing	45.4798	0.0437	0.0222
Postprocessing	0.0093	0.0026	0.0005

6.2. Optimizing Inference Using NPU

The Neural Processing Unit (NPU) is designed for operations related to neural networks. Using the NPU can significantly speed up inference by offloading computations from the CPU and running them in a dedicated hardware block, which is especially useful for real-time processing of large datasets [31].

However, not all models automatically support hardware acceleration, requiring adaptation. Optimization includes replacing unsupported operations with equivalent ones that work efficiently on the NPU and using quantized models to reduce computational load [32].

To use the NPU, follow these steps:

- environment preparation;
- converting the trained model to onnx format;
- quantification of the model in INT8 format, using special HHB tool;
- cross-compilation of the model into an executable program on the CPU/NPU.

This method efficiently processes images, leveraging parallel computing to accelerate embedding extraction.

Using multithreading to distribute computational tasks across CPU cores can improve image processing performance [33]. For example, separate threads can handle preprocessing and inference, allowing them to run un parallel. After using the NPU, good improvements were obtained (Table 4). The launches were carried out on the CPU and NPU.

Table 4. Comparison of data processing time on a neuroprocessor and a central processing unit.

Device	Time using (sec)
NPU	0.063
CPU	67

As one can see from the results, running the model without using an NPU has no practical application, since the execution speed will be too low. Using an NPU significantly speeds up execution.

6. Conclusion

The highest computational load in image processing in our experimental stand comes from the inference and image processing. Therefore, these should be rewritten in C using RVV instructions and, if possible, the NPU accelerator. Using optimized libraries and multithreading can also significantly improve performance.

After applying the NPU, we got quite good improvements (Table 5). For comparison, another YOLOv5n model was used, which is optimized for our NPU. The launches were carried out on the CPU and NPU.

Table 5. Comparing the performance and accuracy of YOLO models on different computing devices.

Model	Device	FPS	Accuracy
YOLOv8n	NPU	5-10	0.967
YOLOv8n	CPU	<1	0.993
YOLOv5n	NPU	>30	0.962
YOLOv5n	CPU	<1	0.991

As one can see from the lest results, the speed increased significantly. Running the model without a NPU does not make sense, since the processing speed will be too low for practical use. But these results do not give us an accurate understanding of whether this board can be used for industrial

applications, since all the steps taken have a lot of pitfalls. However, after solving the problems associated with the development of software tools, it will give a better understanding.

References

[1]. Cui E., Li T. Wei Q. RISC-V instruction set architecture extensions. A survey. IEEE Access 11, 2023, 24696–24711. DOI: 10.1109/ACCESS.2023.3246491.

[2]. Shen Y. Computer Vision: Technologies and Applications. Applied and Computational Engineering, vol. 163, no. 1, pp. 35–41, Jun. 2025, DOI: 10.54254/2755-2721/2025.23817.

[3]. Ali W. Exploring Instruction Set Architectural Variations: x86, ARM, and RISC-V in Compute-Intensive Applications, Aug. 2023, DOI: 10.36227/techrxiv.24026736.

[4]. Han C., Chang C., Srivastava S., Lu Y. Scalable Complex Event Processing on Video Streams. Proc. ACM on Management of Data, vol. 3, no. 3, pp. 1–29, Jun. 2025, DOI: 10.1145/3725419.

[5]. Borysenko V., Borysenko T. Modern approaches of design software applications based on microservice architecture in computer and information systems and technologies. Apr. 2020, DOI: 10.30837/IVcsitic2020201441.

[6]. Bhatnagar S., Mahant R. Designing Microservices in The Art of Decoding Microservices: An In-Depth Exploration of Modern Software Architecture. Launch IT, 2025, pp. 135-192.

[7]. Domenech-Asensi G., Garrigos J., Lopez P., Brea V., Cabello D. Real time architectures for the Scale Invariant Feature Transform algorithm. CNNA 2016; 15th International Workshop on Cellular Nanoscale Networks and their Applications. Dresden, Germany, 2016, pp. 1-2.

[8]. Obukhov A., Dedov D., Volkov A., Rybachok M. Technology for Improving the Accuracy of Predicting the Position and Speed of Human Movement Based on Machine Learning Models. Technologies, vol. 13, no. 3, p. 101, Mar. 2025, DOI: 10.3390/technologies13030101.

[9]. Qin X., Liu X., Han J. A CNN Hardware Accelerator Designed for YOLO Algorithm Based on RISC-V SoC. Proc. IEEE Int. Conf. ASIC, Kunming, China, 2021, pp. 1-4, DOI: 10.1109/ASICON52560.2021.9620500.

[10]. Srivastava S. K, Srivastava A. K., Allam S., Lilaramani D. Comparative analysis on Deep Convolution Neural Network models using Pytorch and OpenCV DNN frameworks for identifying optimum fruit detection solution on RISC-V architecture. IEEE Mysore Sub Section International Conference (MysuruCon), Hassan, India, 2021, pp. 738-743, DOI: 10.1109/MysuruCon52639.2021.9641594.

[11]. Chen Y.-R. Experiments and optimizations for TVM on RISC-V Architectures with P Extension. International Symposium on VLSI Design, Automation and Test (VLSI-DAT), Hsinchu, Taiwan, 2020, pp. 1-4, DOI: 10.1109/VLSI-DAT49148.2020.9196477.

[12]. Yu M.-S., Chang H.-C., Wang C.-T., Tien Y.-W. Optimizing computer vision algorithms with TVM on VLIW architecture based on RVV. The Journal of Supercomputing, vol. 81, no. 1, Nov. 2024, DOI: 10.1007/s11227-024-06530-x.

[13]. Jajal P, Jiang W, Tewari A, Kocinare E, Woo J, Sarraf A. Interoperability in deep learning: a user survey and failure analysis of ONNX model converters. In: Proc. 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM; 2024. p. 1466–1478, DOI: 10.1145/3650212.3680374.

[14]. Fusaomi N., Shingo S., Ryoma A., Keigo W., Maki K. H. Evaluation of Interoperability of CNN Models between MATLAB and Python Environments Using ONNX Runtime Model. AI, Computer Science and Robotics Technology 3(1), 1–13. 2024, DOI: 10.5772/acrt.20240043.

[15]. Sohan M., Ram T. S., Ch V. R. R. A Review on YOLOv8 and Its Advancements. Data Intelligence and Cognitive Informatics, Jan. 2024, pp. 529–545, DOI: 10.1007/978-981-99-7962-2_39.

[16]. Almeyda S., Davila A.: Process Improvement in Software Requirements Engineering: A Systematic Mapping Study. Programming and Computer Software, 48, Aug. 2022, pp. 513–533. DOI: 10.1134/S0361768822080084.

[17]. Lunev D., Poletykin S., Kudryavtsev D. Brain-computer interfaces: Technology overview and modern solutions. Modern Innovations, Systems and Technologies, vol. 2, no. 3, Jul. 2022, pp. 0117-0126, DOI: 10.47813/2782-2818-2022-2-3-01170126.

[18]. Tsekhmystro R., Rubel O., Prysiazhniuk O., Lukin V. V. Impact of distortions in UAV images on quality and accuracy of object localization. radioelectronic and computer systems, Jan. 2025, DOI: 10.32620/reks.2024.4.05.

[19]. Fischer L., Wollstadt P. Precision and Recall Reject Curves for Classification. Aug. 2023, DOI: 10.48550/arXiv.2308.08381.

[20]. Boyd K., Eng K. H., Page C. D. Area under the Precision-Recall Curve: Point Estimates and Confidence Intervals. Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Lecture Notes in Computer Science, vol. 8190, pp. 451–466, Sep. 2013, DOI: 10.1007/978-3-642-40994-3_29.

[21]. Sinha E., Kumar A., Tyagi A. OpenCV for Computer Vision Applications. International Journal For Multidisciplinary Research, vol. 7, no. 3, May 2025, DOI: 10.36948/ijfmr.2025.v07i03.44280.

[22]. Chinnaraju A. Benchmarking cross-platform AI: Web Assembly, ONNX Runtime and TVM for Real-Time Web, Mobile, and IoT Deployment. World Journal of Advanced Research and Reviews, vol. 26, no. 2, pp. 1937–1963, May 2025, DOI: 10.30574/wjarr.2025.26.2.1832.

[23]. Yang S., Lu T. T3 SOC design flow case study: Design a video processing pipeline. ASIC, ASICON '07. 7th International Conference, Nov. 2007, DOI: 10.1109/ICASIC.2007.4415551.

[24]. Jindal K. Design and Implementation of an Embedded Image Processing System on Zynq ZedBoard: A VLSI Perspective. International Journal for Research in Applied Science and Engineering Technology, vol. 13, no. 5, pp. 5141–5145, May 2025, DOI: 10.22214/ijraset.2025.71372.

[25]. Smirnov E., Timoshenko D., Andrianov S. Comparison of Regularization Methods for ImageNet Classification with Deep Convolutional Neural Networks. AASRI Procedia, vol. 6, pp. 89–94, Dec. 2014, DOI: 10.1016/j.aasri.2014.05.013.

[26]. Pujari S. D., Pawar M. M., Wadekar M. Multi-Classification of Breast Histopathological Image Using Xception: Deep Learning with Depthwise Separable Convolutions Model. Techno-Societal, pp. 539–546, May 2021, DOI: 10.1007/978-3-030-69921-5_54.

[27]. Wang S., Wang X., Xu Z., Chen B. Optimizing CNN Computation Using RISC-V Custom Instruction Sets for Edge Platforms. IEEE Trans. Comput, May 2024, pp. 1-14, DOI: 10.1109/TC.2024.3362060.

[28]. Titopoulos V., Alexakis G., Nicopoulos C., Dimitrakopoulos G. Efficient Implementation of RISC-V Vector Permutation Instructions. *arXiv:2505.07112*, May 2025, DOI: 10.48550/arXiv.2505.07112.

[29]. Yuan T., Liu W., Han J., Lombardi F. High Performance CNN Accelerators Based on Hardware and Algorithm Co-Optimization. IEEE Trans. Circuits Syst. I, Reg. Papers, Oct. 2020, pp. 1-14, DOI: 10.1109/TCSI.2020.3030663.

[30]. Jin S., Qi S., Dai Y., Hu Y. Design of Convolutional Neural Network Accelerator Based on RISC-V. Proc. 10th Int. Conf. Appl. Tech. Cyber Intell. (ICATCI 2022), 2023, pp. 446-454. DOI: 10.1007/978-3-031-29097-8_53.

[31]. Cono D'Elia D., Demetrescu C. Ball-Larus Path Profiling across Multiple Loop Iterations. SIGPLAN Not. 48, 10 (oct 2013), pp. 373-390, DOI :10.1145/2544173.2509521.

[32]. Agarwal R., Deshmukh R., Borhade P., Murarka S. Image Classification using Parallel CPU and GPU Computing. Int. J. Eng. Adv. Technol., vol. 9, no. 4, Apr. 2020, pp. 5, DOI: 10.35940/ijeat.D7870.049420.

[33]. Shanthi M., Anthony Irudhayaraj A. Multithreading - An Efficient Technique for Enhancing Application Performance. International Journal of Recent Trends in Engineering, Vol 2, No. 4, Nov. 2009, pp. 165-167, DOI: 10.22146/ijccs.57594.

Информация об авторах / Information about authors

Никита Иванович ЧЕРЕПАНОВ– студент магистратуры высшей школы программной инженерии Санкт-Петербургского политехнического университета Петра Великого. В 2025 получил квалификацию бакалавра в Санкт-Петербургском политехническом университете Петра Великого по специальности "Технология разработки и сопровождения качественного программного продукта". Сфера научных интересов: программные архитектуры, RISC-V, машинное обучение, компьютерное зрение, искусственный интеллект.

Nikita Ivanovich CHEREPANOV is a master's student at the Higher School of Software Engineering at Peter the Great St. Petersburg Polytechnic University. In 2025, he got bachelor degree by graduating from Peter the Great St. Petersburg Polytechnic University with a specialty in "Technology for developing and maintaining a high-quality software product". Research interests: software architectures, RISC-V, machine learning, computer vision, artificial intelligence.

Надежда Олеговна СТЕПИНА – ассистент высшей школы программной инженерии Санкт-Петербургского политехнического университета Петра Великого. В 2023 году окончила Санкт-Петербургский государственный политехнический университет по специальности

«Программная инженерия». В 2024 году стала аспирантом по специальности 05.13.11 – «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей». Область научных интересов – разработка программного обеспечения, машинное обучение, высокопроизводительные вычисления, IoT и embedded-системы.

Nadegda Olegovna STEPINA is an assistant at the Higher School of Software Engineering at Peter the Great St. Petersburg Polytechnic University. In 2023, she graduated from the St. Petersburg State Polytechnic University with a degree in Software Engineering. In 2024, she became a postgraduate student in the field of Mathematical and Software Support for Computing Machines, Complexes, and Computer Networks. Her research interests include software development, machine learning, high-performance computing, IoT, and embedded systems.

Игорь Валерьевич НИКИФОРОВ – доцент высшей школы программной инженерии Санкт-Петербургского политехнического университета Петра Великого. В 2011 году окончил Санкт-Петербургский государственный политехнический университет по специальности «Программное обеспечение вычислительной техники и автоматизированных систем». В 2014 году защитил диссертацию на соискание ученой степени кандидата технических наук по специальности 05.13.11 – «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей». Является автором 100 научных публикаций. Область научных интересов – разработка программного обеспечения, имитационное моделирование, аналитика больших данных, распределенные вычисления.

Igor Valerievich NIKIFOROV. In 2011, he graduated from St. Petersburg State Polytechnic University with a degree in «Computer Science and Automated Systems Software». He got his Cand. Sci. (Tech.) degree in Mathematical and software support for computers, complexes and computer networks in 2014. He is an Associate Professor at the Higher School of Software Engineering at Peter the Great St. Petersburg Polytechnic University. He is the author of more than 100 scientific publications. Research interests – software engineering, simulation modeling, big data analytics, distributed computing.

DOI: 10.15514/ISPRAS-2025-37(5)-13

Generating and Debugging Java Code Using LLMs Based on Associative Recurrent Memory

V.I. Vasilevskiy, ORCID: 0009-0004-0115-7082 <vivasilevskiy_1@edu.hse.ru>
D.V. Alexandrov, ORCID: 0000-0002-9759-8787 <dvalexandrov@hse.ru>
HSE University,
11, Pokrovsky blvd, Moscow, 109028, Russia.

Abstract. Automatic code generation by large language models (LLMs) has achieved significant success, yet it still faces challenges when dealing with complex and large codebases, especially in languages like Java. The limitations of LLM context windows and the complexity of debugging generated code are key obstacles. This paper presents an approach aimed at improving Java code generation and debugging. We propose using the Associative Recurrent Memory Transformer (ARMT) model, which extends the context window and has enhanced memory capabilities, to address two tasks: 1) selecting the most relevant snippets from the existing codebase for generating new code; 2) selecting the most significant parts of stack traces and runtime data for iterative debugging. This approach is integrated with an iterative debugging loop, embodied in our developing system "JavaCapsule" (inspired by PyCapsule for Python), which includes compilation and test execution in a controlled Docker environment using Gradle. It is expected that the proposed method will enhance the accuracy and relevance of generated Java code, particularly in the context of large projects, and improve the automated debugging process. Such benchmarks like JavaBench further underscore the need for such focused advancements. This paper 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).

Keywords: code generation; java; large language models; code debugging; associative recurrent memory transformer; recurrent memory transformer; long context; context selection; iterative debugging; javabench.

For citation: Vasilevskiy V.I., Alexandrov D.V. Generating and Debugging Java Code using LLMs based on Associative Recurrent Memory. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 173-182. DOI: 10.15514/ISPRAS-2025-37(5)-13.

Acknowledgements. This research is conducted at the Cloud and Mobile Technologies Laboratory of the Software Engineering Department at HSE University.

Генерация и отладка Java-кода с использованием больших языковых моделей на основе ассоциативной рекуррентной памяти

В.И. Василевский, ORCID: 0009-0004-0115-7082 <vivasilevskiy_1@edu.hse.ru>
Д.В. Александров, ORCID: 0000-0002-9759-8787 <dvalexandrov@hse.ru>
НИУ ВШЭ,
Россия, 109028, г. Москва, Покровский б-р, д. 11.

Аннотация. Автоматическая генерация кода большими языковыми моделями (LLM) достигла значительных успехов, однако все еще сталкивается с проблемами при работе со сложными и объемными кодовыми базами, особенно на таких языках, как Java. Ограничения контекстного окна LLM и сложность отладки сгенерированного кода являются ключевыми препятствиями. В данной статье представлен подход, направленный на улучшение генерации и отладки Java-кода. Мы предлагаем использовать модель Associative Recurrent Memory Transformer (ARMT), обладающую расширенным контекстным окном и улучшенными возможностями памяти, для решения двух задач: 1) выбора наиболее релевантных фрагментов из существующей кодовой базы для генерации нового кода; 2) выбора наиболее значимых частей стектрейсов и рантаймданных для итеративной отладки. Этот подход интегрирован в итеративный цикл отладки, реализованный в нашей разрабатываемой системе «JavaCapsule» (по аналогии с PyCapsule для Python), которая включает компиляцию и выполнение тестов в контролируемой среде Docker с использованием Gradle. Ожидается, что предложенный метод повысит точность и релевантность генерируемого Java-кода, особенно в контексте крупных проектов, и улучшит процесс автоматизированной отладки. Бенчмарки, такие как JavaBench, дополнительно подчеркивают необходимость подобных целенаправленных усовершенствований.

Ключевые слова: генерация кода; java; большие языковые модели; отладка кода; преобразователь ассоциативной рекуррентной памяти; преобразователь рекуррентной памяти; длинный контекст; выбор контекста; итеративная отладка; оценка моделей javabench.

Для цитирования: Василевский В.И., Александров Д.В. Генерация и отладка Java-кода с использованием больших языковых моделей на основе ассоциативной рекуррентной памяти. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 173–182 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)–13.

Благодарности. Данное исследование ведется в Лаборатории Облачных и Мобильных технологий Департамента Программной Инженерии НИУ ВШЭ.

1. Introduction

Large Language Models (LLMs) demonstrate impressive results in the field of automatic code generation [7]. However, applying these models to complex object-oriented languages like Java presents several difficulties. Java projects are often characterized by large code volumes, complex dependencies, and strong typing, requiring models to have a deep understanding of the context. The JavaBench benchmark [1] has highlighted these challenges, particularly in object-oriented programming (OOP) features and project-level code generation, and underscores the relevance of research in this area.

One of the key problems is the limited context window size of modern LLMs. When generating or modifying a code snippet in a large project, the model must access relevant parts of the existing codebase (other classes, methods, interfaces), which often exceeds the standard context limit. Furthermore, debugging the generated code remains a complex task. Approaches based on analyzing code execution block by block and providing the model with runtime data [3] are quite promising but are limited by the same context window, preventing the transmission of the full stack trace or the history of variable value changes needed to find and fix complex errors.

To address the limited context problem, architectures such as the Recurrent Memory Transformer (RMT) [4] have been proposed, using recurrent mechanisms to process long sequences. A further development of this idea, the Associative Recurrent Memory Transformer (ARMT) [5], adds associative memory, significantly improving memory usage efficiency and performance on long-context tasks, such as BABILong [6].

On the other hand, iterative debugging approaches, where code is executed in a controlled environment (e.g., a container), and the execution results (success/failure, test output) are used for the next generation iteration, have shown effectiveness for Python [2]. We are developing a similar system for Java, named ”JavaCapsule”.

In this study, we propose combining the advantages of ARMT and iterative debugging to create a system for generating and fixing Java code. The main idea is to train an ARMT-like model (or use its attention mechanisms) to select the most relevant information – codebase snippets during generation and parts of the stack trace/state during debugging – which is then passed to the main LLM to perform the task. We hypothesize that such an approach will allow effective work with large Java projects and complex errors, overcoming the limitations of the context window. This research is conducted at the Cloud and Mobile Technologies Laboratory of the Software Engineering Department at HSE University in collaboration with researchers from Huawei Technologies Co. Ltd.

2. Related Work

2.1 Code Generation using LLMs

Significant progress has been made in recent years in using LLMs for code generation (e.g., StarCoder [8], Qwen-Coder [9]). Models are trained on vast code corpora and can generate code from textual descriptions in various languages. However, the quality of generation for complex languages like Java, especially within large projects, requires improvement.

Java Code Generation Benchmarks

The landscape of code generation evaluation has been historically dominated by Python. Recognizing this gap, Cao et al. introduced JavaBench [1], a project-level Java benchmark specifically designed to exercise OOP features. JavaBench comprises four Java projects with 389 methods in 106 classes, featuring high test coverage and attestation by undergraduate students. It aims to address imbalances in programming language focus, code granularity (moving beyond function/statement level), and the lack of testing for advanced OOP features (encapsulation, inheritance, polymorphism) in existing benchmarks. JavaBench’s evaluation design includes multiple context settings and synthesis strategies, providing a more nuanced understanding of LLM capabilities in Java. Its findings emphasize the need for future advancements, especially in providing relevant context like method signatures.

2.2 Long Context Processing

The context window limitation is a fundamental problem for transformers. Various architectures are being developed to address it. RMT [4] introduces recurrence at the segment level using special memory tokens. ARMT [5] improves upon RMT with an associative memory mechanism, demonstrating superiority over RMT and other models like Mamba [10] and RWKV [11] in associative retrieval and ultra-long sequence processing tasks (up to 10 million tokens) on the BABILong benchmark [6]. The BABILong benchmark is specifically designed to evaluate a model’s ability to retrieve and use information distributed across long text, making it relevant for assessing models intended to work with large codebases.

2.3 Code Debugging using LLMs

Automated debugging is another promising direction. LDB (Large Language Model Debugger) [3] proposes using code execution information (execution traces, variable values per basic block) to

identify errors. The model is provided with the execution context, based on which it localizes and fixes the bug. However, as noted, the volume of this information can exceed the context window. PyCapsule [2] implements an iterative approach for Python with two agents (programmer and executor), where code is executed in a Docker container, and test results and compilation errors are used to request corrections from the programmer agent. This approach does not require deep stack trace analysis by the model but may need many iterations. Our JavaCapsule system draws inspiration from this iterative, containerized execution model.

2.4 Relevant Context Selection

The idea of selecting relevant information is not new and is actively used in Retrieval-Augmented Generation (RAG) [12], where an external knowledge base is used to find relevant documents. In the context of code generation, this might mean searching for similar code snippets or documentation. However, initial analysis of solutions like RAG and Repository Mapping indicates they may not be sufficiently effective for selecting precise, deeply-nested contextual information required for complex generation and debugging tasks within large, existing Java codebases. A more refined mechanism is needed, capable of extracting specific dependencies or relevant parts of debugging information. The associative memory mechanisms in ARMT [5] could potentially be adapted to train a model for such selective information extraction from structured context (codebase, stack trace).

3. Proposed Method

Within this research, we are developing ”JavaCapsule”, a system for Java code generation and debugging based on the following components:

1. Context/Debugging Selection Model based on ARMT: The core of the system is a model utilizing ARMT principles, trained to perform two main functions:
   - Code Context Selection: Upon receiving a request for Java code generation or modification (e.g., description of a method, class to be changed), the model analyzes the current codebase (provided as an indexed set of files or a structural representation) and selects the most relevant snippets (imports, signatures of other methods, class fields, parent classes, interfaces) necessary for correct generation. The associative memory mechanism of ARMT [5] can be used to establish connections between the request and relevant code sections.
   - Debugging Information Selection: When a compilation error or test failure occurs, the model receives the error message and stack trace. The model’s task is to select the most informative lines or blocks from the stack trace and possibly from the execution history (if available) that indicate the cause of the problem. This allows focusing the LLM on the source of the error without overloading the context with redundant information.
2. Main LLM Generator: We have selected ‘gemma3-27b-it‘ as the base large language model capable of generating Java code. It receives the original user request as input, augmented with the relevant code context or debugging information selected by the model in step 1.
3. Iterative Debugging Loop (JavaCapsule Workflow): The generation and debugging process is iterative, adapting the idea from [2]:
   - User Request: A user, potentially through an IDE plugin, submits a task description for code generation or modification.
   - Generation: The main LLM (‘gemma3-27b-it‘) generates Java code based on the request and context selected by the ARMT-based model (if applicable for an existing project).

- Compilation and Testing (Execution Agent): The generated code (or modified project) is passed to an Execution Agent. This agent compiles the Java code using standard tools (e.g., Gradle) and runs user-provided or automatically generated unit tests (e.g., JUnit). This step is performed in an isolated Docker container for security, dependency management, and reproducibility.
- Result Processing:
  - Successful Execution: If compilation and all tests pass, the final code is returned to the user (e.g., displayed in the IDE).
  - Compilation Error: If a compilation error occurs, the error logs are captured by the Execution Agent. These logs are then processed by the ARMT-based debugging information selection model to extract relevant error messages. The selected information is passed back to the main LLM along with the problematic code for a correction attempt.
  - Test Failure: If tests fail, the Execution Agent runs tests in debug mode to gather more comprehensive debug data (e.g., stack traces, intermediate variable values if feasible). This complete debug data is passed to the ARMT-based model, which selects the most relevant context from this data and the original code. The problematic code, selected debug context, and test error information are then sent to the main LLM for fixing.
- Repetition: The loop (Fixed code -> Retry with new code) repeats from the Compilation and Testing step until successful execution or an iteration limit is reached.

The architecture, inspired by PyCapsule [2], is shown in Fig. 1. The key distinction in our JavaCapsule approach is the explicit use of an ARMT-based model for intelligent selection of both code context during generation and specific debugging information during the error-fixing iterations.

A key aspect is training the ARMT model (or its equivalent) for selection tasks. This requires creating specialized datasets for Java, where generation requests are annotated with relevant parts of the codebase, and error reports are annotated with relevant stack trace lines.

4. Preliminary Considerations and Future Work

This research is currently in the concept development and initial implementation stage. The JavaCapsule system (repository: https://github.com/Vvil1568/JavaCapsule) is under active development, aiming to realize the iterative debugging loop with Docker and Gradle for Java.

Before settling on the ARMT-based selection model, several alternative approaches were considered. One involved using a 7B model where input tokens are replaced by pre-computed BERT embeddings for Java code snippets. Another approach was to fine-tune a 7B model using ParameterEfficient Fine-Tuning (PEFT) techniques like LoRA to generate Java code with explicit type annotations (e.g., transforming ‘someObj.foo()‘ into ‘((SomeClass)someObj).foo()‘). While promising, these approaches primarily focus on altering the generation process itself. The ARMT approach was ultimately chosen for its direct focus on solving the core problem of long-context retrieval and relevance filtering, which we believe is a more fundamental bottleneck for working with large codebases and complex debugging scenarios. A comparative computational cost analysis of these three plans is provided in the next section. Expected Advantages:

- Improved quality of Java code generation for large projects by providing the LLM with relevant context.
- Enhanced automated debugging by focusing the LLM on significant parts of diagnostic information.
- Overcoming LLM context window limitations without needing models with ultra-large context for the entire task.

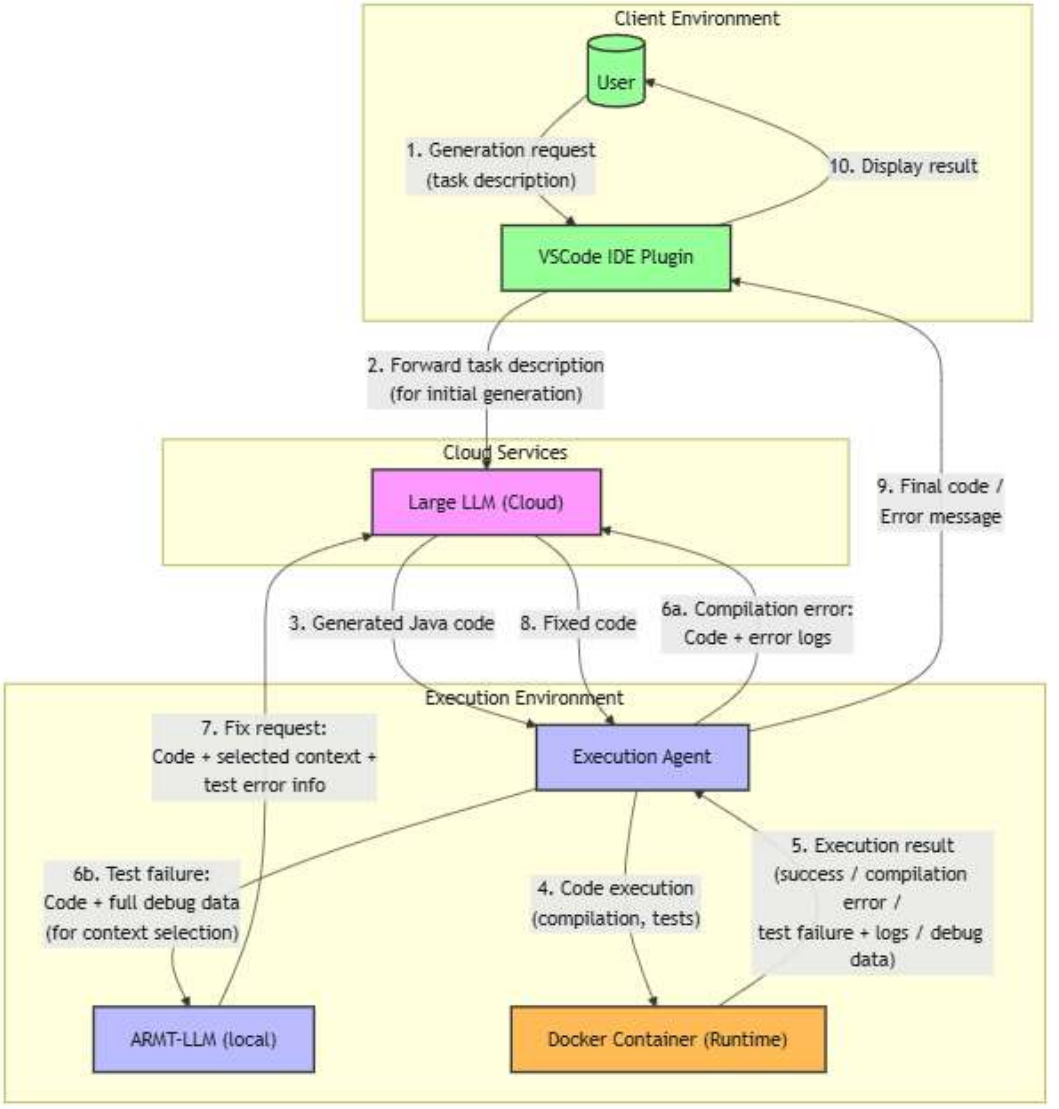


Fig. 1. Adapted architecture for iterative Java code generation and debugging in JavaCapsule. Steps include generation by an LLM (e.g., Gemma), execution/testing in a Docker container via Gradle, and a feedback loop using an ARMT model for context/debugging information selection (adapted from [2]). The "ARMT-LLM (local)" in the diagram represents our proposed ARMT-based selection model.

Main Challenges:

- Training Data Creation: Developing methodologies and tools for creating datasets linking generation/debugging requests with relevant code/stack trace snippets in Java projects. We plan to initiate dataset collection, potentially using automated labeling with existing generative models as a starting point.
- Training the Selection Model: Choosing the architecture (ARMT or similar) and effectively training the model for selective information extraction from structured, yet large, contexts.
- Component Integration: Creating an efficient pipeline combining the selection model, the LLM generator ('gemma3-27b-it'), and the JavaCapsule compilation/testing system.

- Evaluation: The JavaBench benchmark [1] provides a valuable resource. We have started adapting JavaBench for evaluating the Gemma model (adapted version available at: https://github.com/Vvil1568/JavaBench/tree/gemma). This will require further development of custom metrics and test scenarios for our specific tasks.

Future Work Plan:

1. Continue development of the JavaCapsule prototype system for compiling and testing Java code in a container using Gradle.
2. Begin creation of a dataset for training and evaluating the code context selection model using several open-source Java projects.
3. Initiate creation of a dataset for training and evaluating the debugging information selection model based on real or synthetic errors, exploring automated labeling techniques.
4. Experiment with Gemma-3 models as the generator and as the base for the ARMT selector.
5. Compare the proposed approach with baseline LLMs (without context selection) and standard RAG approaches on the adapted JavaBench and custom tasks.
6. Evaluate performance on tasks such as generating new methods, modifying existing code, and fixing errors based on tests within the JavaCapsule framework.

We plan to use code quality metrics (e.g., Pass@k test pass rate from JavaBench, CodeBLEU [13]) and debugging efficiency metrics (number of iterations, percentage of fixed errors).

5. Computational Cost Analysis of Training Approaches

This section provides a high-level estimation of the computational resources required for training the models under the three considered plans. These figures are approximate and intended to provide a sense of scale for each approach. A summary is presented in Table 1.

5.1 General Assumptions

The following assumptions are used for the calculations:

- Precision: BF16 (2 bytes per parameter).
- Optimizer: AdamW, which requires 2 additional values per parameter, resulting in 4 bytes per parameter for optimizer states. Gradients require 2 bytes/param.
- Total Memory per Parameter (Full Training): 2 (params) + 4 (optimizer) + 2 (gradients) = 8 bytes. For PEFT, this applies only to trainable parameters.
- Total FLOPs (heuristic): Approximately $6\times N \times D$ for full training and $\approx 3\times N_{full} \times D$ for PEFT, where N is the number of parameters and D is the number of tokens.
- Model FLOPs Utilization (MFU): We assume a realistic MFU of 40% of the GPU's peak theoretical performance.

5.2 Plan 1: 1.1B ARMT-based Context Selector (Full Tuning)

- Model and Task: A 1.1B parameter ARMT-like model for context selection, trained via full fine-tuning.
- Training Data: Estimated at 40 billion tokens.
- VRAM (Video Memory) Estimation:
  - Model States: 1.1×109 params ×8 bytes/param = 8.8 GB.
  - Activations: Can consume 15-30 GB or more, depending on batch size and sequence length.
  - Total Estimated VRAM: 24-44+ GB.

- Training Time Estimation:
  - Total TFLOPS Required: 6×(1.1×109)×(40×109) = 2.64×1020 FLOPS, or 264,000 PetaFLOPS.
  - Estimated GPU-Hours (H100):

$$\frac{2.64 \times 10^{20}}{1000 \times 10^{12} \times 0.4 \times 3600} \approx 183.300\ GPU - hours$$

5.3 Plan 2: 7B Model with BERT Embeddings (Full Tuning)

- Model and Task: A 7B parameter model for code generation, where input tokens are precomputed BERT embeddings. This involves full fine-tuning.
- Training Data: Estimated at 15 billion BERT-vector ”tokens”.
- VRAM (Video Memory) Estimation:
  - Model States: 7×109 params ×8 bytes/param = 56 GB.
  - Activations: For a 7B model, activations can easily require 20-40+ GB.
  - Total Estimated VRAM: 56 GB + (20-40+ GB) = 76-96+ GB. This requires A100 (80GB) or H100 GPUs.
- Training Time Estimation:
  - Total TFLOPS Required: 6×(7×109)×(15×109) = 6.3×1020 FLOPS, or 630,000 PetaFLOPS.
  - Estimated GPU-Hours (H100):

$$\frac{6.3 \times 10^{20}}{1000 \times 10^{12} \times 0.4 \times 3600} \approx 437.500\ GPU - hours$$

5.4 Plan 3: 7B Model with Explicit Types (PEFT)

- Model and Task: Fine-tuning a 7B parameter model using PEFT (LoRA) to generate Java code with explicit type annotations.
- Training Data: Requires a large dataset of Java code pre-processed with explicit types, estimated at 100 billion tokens.
- VRAM (Video Memory) Estimation:
  - Frozen Model: 7×109 params ×2 bytes/param = 14 GB.
  - LoRA Adapters (70M params): Optimizer states and gradients for adapters require 70×106×6 bytes ≈ 0.42 GB.
  - Activations: Calculated for the full 7B model, requiring 20-40+ GB.
  - Total Estimated VRAM: 14 GB + 0.5 GB + (20-40+ GB) = 35-55+ GB. Suitable for 48GB-class GPUs and above.
- Training Time Estimation:
  - Total TFLOPS Required: $3 \times (7 \times 109) \times (100 \times 109) = 2.1 \times 1021$ FLOPS, or 2,100,000 PetaFLOPS.
  - Estimated GPU-Hours (H100):

$$\frac{2.1 \times 10^{21}}{1000 \times 10^{12} \times 0.4 \times 3600} \approx 1{,}458.000\ GPU - hours$$

These estimates underscore that all considered plans require access to significant high-performance computing infrastructure.

Table 1. Summary table of computational cost estimates for different training approaches.

Plan	Parameters (Training)	VRAM (Estimate)	Tokens (Training)	TFLOPS (Total)	GPU-Hours (H100, 40% MFU)	Suitable GPUs (VRAM)
1. 1B+ARMT (context selection)	1.1B (full)	24-44+ GB	40 billion	264,000 PFLOPs	183.300	4090(48GB), A100, H100
2. 7B+BERT emb (generation)	7B (full)	76-96+ GB	15 billion (BERT)	630,000 PFLOPs	437.500	A100(80GB), H100
3. 7B+Types (generation, PEFT)	7B (70M PEFT)	35-55+ GB	100 billion	2,100,000 PFLOPs	1,458.000	4090(48GB), A100, H100

6. Conclusion

Generating and debugging code for complex, large Java projects using LLMs presents a current and unresolved challenge. Context window limitations and the difficulty of interpreting the entire codebase or full stack traces are significant constraints. In this research, we propose a novel approach based on using an Associative Recurrent Memory Transformer (ARMT) type model for intelligent selection of relevant code context and diagnostic information. This selected information is then passed to a main LLM (Gemma-3-27b-it) for code generation or correction within an iterative loop involving compilation and testing, embodied in our developing JavaCapsule system. The adaptation of benchmarks like JavaBench will be crucial for evaluation. We expect this approach to enhance the accuracy, relevance, and efficiency of automated Java code generation and debugging, opening new possibilities for applying LLMs in enterprise-level software development. Future work will focus on dataset creation, training the selection model, and experimentally validating the proposed system.

References

[1]. Cao J., Chen Z., Wu J., Cheung S., Xu C. JavaBench: A Benchmark of Object-Oriented Code Generation for Evaluating Large Language Models. arXiv preprint arXiv:2406.12902, 2024.
[2]. Adnan M., Xu Z., Kuhn C. C. N. Large Language Model Guided Self-Debugging Code Generation. arXiv preprint arXiv:2502.02928, 2025.
[3]. Zhong L., Wang Z., Shang J. LDB: A Large Language Model Debugger via Verifying Runtime Execution Step by Step. arXiv preprint arXiv:2402.16906, 2024.
[4]. Bulatov A., Kuratov Y., Burtsev M. S. Recurrent memory transformer. Advances in Neural Information Processing Systems, vol. 35, 2022, pp. 11079-11091.
[5]. Rodkin I., Kuratov Y., Bulatov A., Burtsev M. Associative Recurrent Memory Transformer. In Proc. of the ICML 2024 Next Generation of Sequence Modeling Architectures Workshop, 2024.
[6]. Kuratov Y., Bulatov A., Anokhin P., Rodkin I., Sorokin D., Sorokin A., Burtsev M. BABILong: Testing the Limits of LLMs with Long Context Reasoning-in-a-Haystack. arXiv preprint arXiv:2406.10149, 2024.
[7]. Chen M., Tworek J., Jun H., Yuan Q., Pinto H. P. D. O., Kaplan J., ... Brockman G. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.
[8]. Li R., Allal L. B., Zi Y., Muennighoff N., Kocetkov D., Mou C., ... Li J. Starcoder: may the source be with you! arXiv preprint arXiv:2305.06161, 2023.
[9]. Hui B., Yang J., Cui Z., Yang J., Liu D., Zhang L., ... Lin J. Qwen2. 5-Coder Technical Report. arXiv preprint arXiv:2409.12186, 2024.

[10]. Gu A., Dao T. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. arXiv preprint arXiv:2312.00752, 2023.
[11]. Peng B., Alcaide E., Anthony Q., Albalak A., Arcadinho S., Cao H., ... Zhu R. J. RWKV: Reinventing RNNs for the Transformer Era. arXiv preprint arXiv:2305.13048, 2023.
[12]. Lewis P., Perez E., Piktus A., Petroni F., Karpukhin V., Goyal N., ... Kiela D. Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 9459-9474.
[13]. Ren S., Zhou D., Zhang S., Liu S., Chen Y., Sun H., ... Liu Y. CodeBLEU: a method for automatic evaluation of code synthesis. arXiv preprint arXiv:2009.10297, 2020.

Информация об авторах / Information about authors

Владимир Игоревич ВАСИЛЕВСКИЙ – стажер-исследователь Лаборатории Облачных и Мобильных Технологий Факультета Компьютерных Наук НИУ ВШЭ. Сфера научных интересов: большие языковые модели, генерация и отладка кода, обработка длинных последовательностей, компиляторы.

Vladimir Igorevich VASILEVSKIY is a research assistant at the Cloud and Mobile Technologies Laboratory of the Faculty of Computer Science, HSE University. His research interests include large language models, code generation and debugging, long sequence processing, and compilers.

Дмитрий Владимирович АЛЕКСАНДРОВ – профессор департамента программной инженерии факультета компьютерных наук НИУ “Высшая школа экономики”, заведующий научно-учебной лаборатории облачных и мобильных технологий. Сфера научных интересов: методы и технологии искусственного интеллекта, машинное обучение и анализ данных, iOS разработка, разработка мобильных приложений, разработка программного обеспечения, indoor навигация, базы данных, разработка игр.

Dmitry Vladimirovich ALEXANDROV is a Professor in the Department of Software Engineering, Faculty of Computer Science, National Research University “Higher School of Economics”. He is also the Head of the Research and Educational Laboratory of Cloud and Mobile Technologies. His research interests include methods and technologies of artificial intelligence, machine learning and data analysis, iOS development, mobile application development, software development, indoor navigation, databases, game development.

DOI: 10.15514/ISPRAS-2025-37(5)-14

Modification of the Smith-Waterman Algorithm for Local Alignment of Genetic Sequences Based on the Window Method

E.S. Bezuglova, ORCID: 0000-0002-7608-0452 <bezuglova.ketrin@yandex.ru>
E.M. Shiriaev, ORCID: 0000-0002-2359-1291 <egor.shiriaev@reaneling.ru>
N.N. Kucherov, ORCID: 0000-0003-0337-0093 <nik.bekesh@yandex.ru>
M.G. Babenko, ORCID: 0000-0001-7066-0061 <whbear@yandex.ru>

North-Caucasus Federal University, Faculty of Mathematics and Computer Science named after Professor N.I. Chervyakov, 1, Pushkin st., Stavropol, 355017, Russia.

Abstract. The paper presents a modified algorithm for local alignment of genetic sequences based on the Smith-Waterman algorithm, using window method and run-length encoding. an experimental comparison of the performance of the proposed approach with the classical algorithm by such metrics as execution time, average and peak memory usage is carried out. The results demonstrate the effectiveness of the modification while preserving the quality of alignment, especially in resource-constrained environments. The work has practical implications for bioinformatics tasks involving genome analysis, gene annotation and homologous site search.

Keywords: Smith-Waterman algorithm; run-length encoding; window method; genetic sequences; local alignment; bioinformatics.

For citation: Bezuglova E.S., Shiriaev E.M., Kucherov N.N., Babenko M.G. Modification of the Smith-Waterman algorithm for local alignment of genetic sequences based on the window method. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 183-194. DOI: 10.15514/ISPRAS-2025-37(5)-14.

Acknowledgements. The research was supported by the Russian Science Foundation Grant No 25-71-30007, https://rscf.ru/en/project/25-71-30007/.

Модификация алгоритма Смита-Ватермана для локального выравнивания генетических последовательностей на основе метода окна

Е.С. Безуглова, ORCID: 0000-0002-7608-0452 <bezuglova.ketrin@yandex.ru>
Е.М. Ширяев, ORCID: 0000-0002-2359-1291 <egor.shiriaev@reaneling.ru>
Н.Н. Кучеров, ORCID: 0000-0003-0337-0093 <nik.bekesh@yandex.ru>
М.Г. Бабенко, ORCID: 0000-0001-7066-0061 <whbear@yandex.ru>

Северо-Кавказский федеральный университет,
факультет математики и компьютерных наук имени профессора Н.И. Червякова,
Россия, 355017, Ставрополь, Пушкина, д. 1.

Аннотация. В статье представлен модифицированный алгоритм локального выравнивания генетических последовательностей, основанный на алгоритме Смита-Ватермана, с использованием метода окон и кодирования длин серий. Проведено экспериментальное сравнение производительности предлагаемого подхода с классическим алгоритмом по таким метрикам, как время выполнения, среднее и пиковое потребление памяти. Результаты демонстрируют эффективность модификации при сохранении качества выравнивания, особенно в условиях ограниченных вычислительных ресурсов. Работа имеет практическое значение для задач биоинформатики, связанных с анализом геномов, аннотированием генов и поиском гомологичных участков.

Ключевые слова: алгоритм Смита-Ватермана; кодирование длин серий; метод окон; генетические последовательности; локальное выравнивание; биоинформатика.

Для цитирования: Безуглова Е.С., Ширяев Е.М., Кучеров Н.Н., Бабенко М.Г. Модификация алгоритма Смита-Ватермана для локального выравнивания генетических последовательностей на основе метода окна. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 183–194 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)–14.

Благодарности. Исследование выполнено за счет гранта Российского научного фонда № 25-71-30007, https://rscf.ru/project/25-71-30007/.

1. Introduction

Genetic sequence alignment is a bioinformatics technique that is based on placing two or more sequences of deoxyribonucleic acid (DNA), ribonucleic acid (RNA) or protein monomers under each other in such a way that similar sequence regions can be observed to understand functional, structural and evolutionary relationships [1].

The application of biological sequence alignment methods played a key role in one of the most significant projects of modern science - the Human Genome Project [2], in which the results of alignment formed the basis for subsequent functional annotation of genes, identification of genetic variants associated with diseases, and development of personalized medicine.

Among the local alignment algorithms, the most popular is the Smith-Waterman algorithm [3], which provides accurate matching of sequence fragments through dynamic programming. Despite the high accuracy of this algorithm, its classical representation has significant computational costs, namely, its time and space complexity is $O(n \times m)$, where n and m are the lengths of the compared sequences. This limits its applicability when analysing long sequences or in conditions of limited computational resources (e.g., on lined systems or in fog computing environments).

Current approaches in algorithm optimization focus on making use of high-performance computing. These include vectorization using single instruction, multiple data (SIMD) principles [4-5], parallel implementations on GPUs [6-7], and the use of specialized architectures [8]. Despite the efficiency of the above approaches, the solutions require additional hardware and are not always applicable in distributed computing environments.

The actual task is to develop compact and efficient algorithms that do not depend on the architectural features of the computing platform.

In the framework of the proposed study, a modification of the classical Smith-Waterman local alignment algorithm is implemented using several known, but previously unused, joint techniques: binarization of global sequences, data compression using the run-length encoding (RLE) method [9], and partitioning data into fixed windows. Although the use of this method separately is widely used in computational biology and information technology, the presented combination and its practical application to local competition problems are constantly new.

The main novelty and originality of the proposed solution are as follows:

- initials and names of the authors;
- an original scheme for binarization of nucleotides sequentially is proposed, which allows for a significant reduction in the volume of original data. In combination with RLE encoding of binary data, high compression efficiency is achieved, especially in areas with repeating symbols typical of genomic sequences;
- integration of binary representation and RLE encoding allows for speeding up the process of sequence comparison, since the competition is reduced to bit operations instead of symbol-by-symbol comparisons. This, in turn, provides additional performance gains, especially in conditions of limited computing resources;
- the use of fixed windows for data processing provides not only a further reduction in computational costs, but also the possibility of efficient implementation of the algorithm in parallel environments and the distribution of distributed environments, such as systems with limited computing power or fog computing.

The paper proposes a modification of the Smith-Waterman algorithm based on the preliminary binary representation of sequences followed by bitwise compression based on RLE. This approach allows to significantly reduce the amount of processed data and speed up computations by simplifying the matching operations. In addition, an adapted penalty system for the binary format is introduced to preserve the algorithm's sensitivity to biologically significant changes.

2. Method

2.1 Data presentation

To reduce the data volume of the processed information and increase the efficiency of local alignment operations, the proposed algorithm first performs the conversion of nucleotide sequences into a binary data representation [10]. This conversion is based on the binary encoding of each nucleotide using a two-bit scheme, which provides a compact and convenient representation of the input data.

In the proposed approach, a unique two-bit combination is assigned to each symbol of a nucleotide sequence from the set $\{A, C, T, G\}$:

$$A \to 00, C \to 01, T \to 10, G \to 11.$$

This scheme provides efficient memory usage because it allows any sequence of length n to be represented as a bit string of length $2n$ bits, which halves the character representation compared to a character representation using 8 bits per character.

The conversion of the character string to a binary string is implemented by the function «dna_to_binary(dna_str)», which replaces each nucleotide with the advised 2-bit string. The reverse conversion is implemented by the function «binary_to_dna(binary_str)» and is necessary to restore the interpreted result after calculations in binary representation.

This conversion allows bitwise comparisons instead of character comparisons, which significantly speeds up the work of the modified algorithm. Instead of string operations comparing two nucleotides (for example, $A == G$), an exclusive-or operation (XOR) [11] is performed between 2-bit codes, and the result is 0 only if there is a complete match. This approach is easily scalable for vector computations and hardware optimisations including the SIMD principle. The described contributions are presented in Algorithm 1.

Algorithm №1 Function for representing a sequence in binary form

Input: dna_str – string of characters $\{A, C, T, G\}$
Output: $binary\_str$ – string of characters $\{0, 1\}$

1. $bin\_map \leftarrow \{'A': '00', 'C': '01', 'T': '10', 'G': '11'\}$
2. $binary\_str \leftarrow empty\ string$
3. For i from 0 to length $(dna\_str) - 1$ is executed
3.1. $nucleotide \leftarrow dna\_str[i]$
3.2. $bin\_str \leftarrow binary\_str + bin\_map[nucleotide]$
3.3. End of cycle
4. Return bin_str
5. Define $dna\_map \leftarrow \{'00': 'A', '01': 'C', '10': 'T', '11': 'G'\}$
6. If $length(bin_{str}) mod\ 2 \neq 0$ then
7. $bin\_str \leftarrow bin\_str + '0'$
8. For i from 0 to $length(bin\_str) - 1\ step\ 2$ do
8.1. $bits \leftarrow bin\_str[i] + bin\_str[i+1]$
8.2. $nucleotide \leftarrow dna\_map[bits]$
8.3. $dna\_str \leftarrow dna\_str + nucleotide$
8.4. End of cycle
9. Return dna_string

2.2 Application of compression

The After converting the nucleotide sequence into a binary representation, data compression is performed. It is needed in order to reduce redundancy and reduce the amount of information that needs to be equalized. The repetitive sequence encoding algorithm, RLE, has been used for this purpose.

The RLE method is one of the simplest and least resource-intensive methods of data compression, in which sequences consisting of identical characters (in this case – bits) are replaced by a pair of values: character and number of repetitions. For example, the substring 000001111 will be encoded as 05 14, which will reduce the total amount of representation in the presence of long single-type data blocks.

When representing genetic data in binary form, the proposed method shows high efficiency, because binary strings, binary strings derived from nucleotide sequences, often contain repetitive fragments, for example, homopolymer regions – sequences like AAAA, CCCC, etc. The binary format of genetic data shows high efficiency. In addition, the binary format itself has low entropy compared to the character format, which further enhances the efficiency of RLE coding [12].

The following factors explain the choice of compression technique:

- the RLE algorithm has a low computational complexity of $O(n)$, so it is suitable for processing large data sets without significant computational cost [13];
- window method compression, where a string in binary representation is split into fixed-length fragments, each of which is compressed separately. These actions make the algorithm robust to local changes and decoding with minimal resource consumption;

- unlike more complex methods (e.g., Huffman's algorithm [14] or arithmetic coding [15]), recovering data from an RLE representation does not require the construction of external tables and can be performed in a single linear pass, which is especially important in resource-constrained environments such as embedded systems or fog computing nodes [16];
- after applying RLE, additional data compression is possible, providing multi-layer compression without data loss.

The Huffman algorithm [14, 17] and arithmetic coding [15, 18] are used to efficiently encode character sequences, especially in the presence of statistical differences in character frequencies, but they require the construction and storage of auxiliary structures, which increases the amount of metadata and complicates decoding.

The compression methods LZ7 and LZ78 [19-20] and their derived algorithms, including the Lempel-Ziv-Welch algorithm (LZW) and the Lempel-Ziv-Markov chain algorithm (LZMA) [21], show a high compression rate when processing large amounts of text data [19], but they also require large buffers and time for preliminary analysis, which makes them less suitable for low-level optimization in a limited computational environment.

RLE, in turn, has high speed and minimal memory requirements, which makes it particularly efficient in the presence of patterned, repetitive patterns - exactly the kind of structures characteristic of genetic sequences. In addition, compared to Huffman coding and LZW, RLE shows better performance on short fragments and in applications with «real-time» data processing [22].

In the proposed approach, local alignment is not performed at the nucleotide level, but in the space of binary strings that were obtained by performing binary coding.

The splitting into windows is performed using the function «split_into_windows», after which separate processing is performed. Each window is compressed using the RLE method. The data recovery process allows to exactly restore the original window content and, if necessary, to perform an accurate restoration of the binary sequence (Algorithm 2).

An important aspect of the proposed approach is that binarization and subsequent RLE coding of genetic sequences do not impose restrictions on the ability to identify alignments of arbitrary length or location. Despite the fact that the local alignment algorithm is performed in binary representation, this is an exclusively intermediate form of data that ensures efficient processing. After the alignment procedure is completed, an exact reverse transformation from the binary representation to the original nucleotide sequences is implemented. This ensures complete compliance of the alignment results with the original sequences without loss of information accuracy. Thus, any alignments found at the binary level are unambiguously and accurately translated back into the original nucleotide sequences, preserving the biological significance and accuracy of the final result.

2.3 Window method

Window method is a commonly used technique in sequence, signal, and text processing algorithms, where the input data is partitioned into fixed-size non-overlapping segments (windows) [23]. Unlike sliding window techniques, each window in this method is processed independently without overlap. In the context of biological data processing, particularly sequence alignment, the window method improves computational efficiency by reducing the working data set size. This approach also facilitates parallel execution, as each window can be processed independently, which enhances performance and optimizes memory usage.

In the proposed window method, S is a binary string of length n and the window size is W. The string S is partitioned into windows, each consisting of a sequence of length W, except for the outermost window which can be shorter if the length of string n is not divisible by W.

Algorithm №2 Data compression and recovery

Input: $binary\_str, window\_size$

Output: $compressed\_windows, decompressed\_windows$

1. $windows \leftarrow empty\ list$
2. For i from 0 to length ($binary\_str$) step $window\_size$ do
2.1. $window \leftarrow substring$ of $binary\_str$ for index i to $i + window\_size$
2.2. Append $window$ to $windows$
3. End of cycle
4. $compressed\_windows \leftarrow empty\ list$
5. For each $window$ in $windows$ do
5.1. If $window$ is empty, then continue
5.2. $compressed \leftarrow empty\ string$
5.3. $current\_bit \leftarrow first\ bit\ of\ window$
5.4 $count \leftarrow 1$
5.6 For bit in $window$ starting from second position do
5.6.1. If $bit = current\_bit$, then
5.6.1.1. $count \leftarrow count + 1$
5.6.2. Else
5.6.2.1. Append $current\_bit + count$ to $compressed$
5.6.2.2. $current\_bit \leftarrow bit$
5.6.2.3. $count \leftarrow 1$
5.7 Append final $current\_bit + count$ to $compressed$
5.8. Append $compressed$ to $compressed\_windows$
6. End of cycle
7. $decompressed\_windows \leftarrow empty\ list$
8. For each $compressed\_win$ in $compressed\_windows$ do
8.1. $window \leftarrow empty\ string$
8.2. i $\leftarrow 0$
8.3. While $i < length(compressed\_win)$ do
8.3.1. $bit \leftarrow compressed\_win[i]$
8.3.2. $count \leftarrow integer\ value\ of\ compressed\_win[i+1]$
8.3.3. Append $bit * count$ to $window$
8.3.4. $i \leftarrow i + 2$
8.4. Append $window$ to $decompressed\_windows$
9. End of cycle
10. Return $compressed\_windows, decompressed\_windows$

The process of partitioning into windows:

$$windows(S, W) = [S[i: i + W] | i = \{0, W, 2W, \dots, \left\lfloor (\frac{n-W}{W} \right\rfloor \times W\}],$$

where i is the start index of each window, W is the window size, and $S[i: i + W]$ is a slice of the string S starting at index i and of length W. The expression $\left\lfloor (\frac{n-W}{W} \right\rfloor \times W$ ensures that no window exceeds the bounds of the sequence.

If the string length n is not a multiple of W, the last window will have a shorter length equal to $n\ mod\ W$, where $n\ mod\ W$ is the remainder of dividing n by W. The partitioning algorithm ensures that the entire string is covered by non-overlapping windows, with the last one possibly shorter than the others.

Partitioning a string into smaller, fixed-size windows allows for the distribution of computational tasks, as each window can be processed independently. This feature is particularly beneficial in

multitasking or multiprocessor systems, where each window can be assigned to a separate thread or process. The technique also contributes to optimized memory usage, enabling processing of smaller data portions without requiring the entire sequence to be loaded into memory.

Thus, the described algorithms can be implemented as a unified program or used independently in unrelated tasks. Based on these modules, a working program has been developed, and its results are presented in the next section.

3. Conducting the experiment

To evaluate the efficiency of the proposed algorithm for local alignment of genetic sequences, a series of computational experiments were conducted.

The study was carried out on the basis of programmes described in the Python programming language [24], on a dataset of genetic sequences ranging from 100 to 3000 nucleotides, with a step of 100, on equipment with the following characteristics:

- CPU: Apple M2 16 cores, 3.4GHz;
- RAM: 8GB LPDDR4X;
- Storage: 512GB NVMe SSD PCIe 3.0;
- Operating System: macOS Sonoma 15.4.

The experiment was conducted in three phases:

1) measurement of speed, average memory and peak memory when performing local alignment using the classical Smith-Waterman algorithm;
2) measurement of speed, average memory and peak memory when performing local alignment using the modified algorithm based on the Smith-Waterman algorithm;
3) comparisons of the obtained results.

Average memory usage was computed as the mean value of allocated memory during the entire execution, while peak memory corresponds to the maximum memory usage observed at any point. These metrics were chosen to reflect both sustained and worst-case memory behavior.

The results showed that as sequence length increased from 100 to 3000 nucleotides, the classical Smith-Waterman algorithm exhibited a quadratic increase in execution time (up to 520 seconds). In contrast, the modified version maintained a nearly constant execution time between 10 and 20 seconds. This demonstrates a significant reduction in computational load – approximately 11.7-fold (Fig. 1).

Fig. 2 shows a comparison of the average memory consumption between the classical and modified algorithms as a function of sequence length. The results demonstrate that the classical algorithm exhibits linear growth in average memory usage as the input sequence length increases, exceeding 20,000 KB for sequences of 3,000 nucleotides.

In contrast, the modified algorithm displays a more stable behavior: after a brief initial rise, average memory usage stabilizes at approximately 3,800 KB and remains nearly constant regardless of further increases in input size.

These findings confirm that the proposed method is particularly effective for use in systems with limited computational resources, where predictable and low memory usage is critical.

Fig. 3 presents a comparison of peak memory consumption between the classical and modified algorithms. The classical Smith-Waterman algorithm is characterized by an exponential increase in peak memory usage as sequence length grows, reaching approximately 72,088 KB for input sequences of 3,000 nucleotides.

In contrast, the modified algorithm shows significantly lower peak memory usage: after a moderate initial increase, the values stabilize around 7.8 MB, regardless of further input growth.

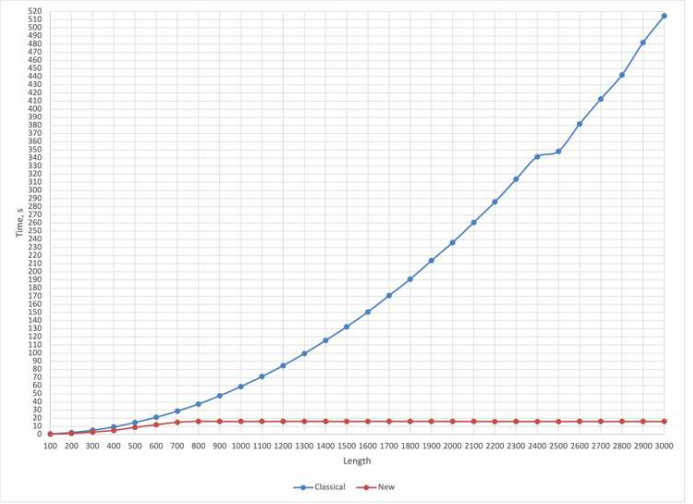


Fig. 1. Comparison of average time used

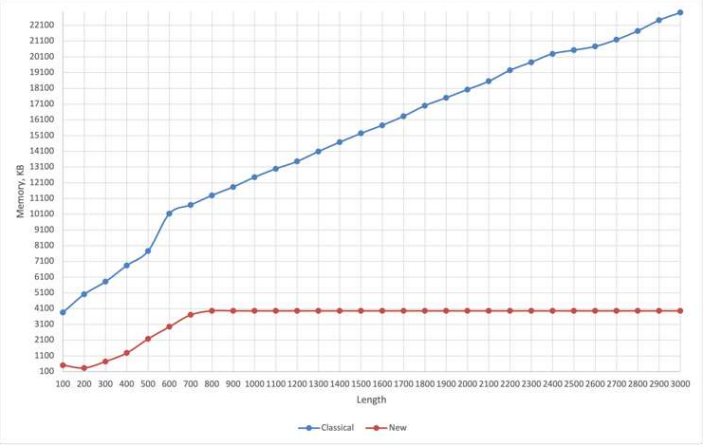


Fig. 2. Comparison of average memory consumption

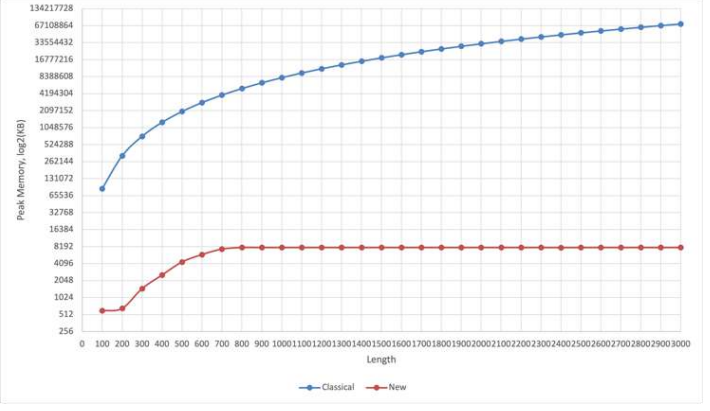


Fig. 3. Comparison of peak memory consumption

These results confirm the effectiveness of the proposed window-based compression and processing technique, which substantially reduces memory load while preserving the accuracy of the alignment.

4. Conclusion

The paper proposes an algorithm for local alignment of genetic sequences based on window method and RLE compression. The results of computational experiments show a significant reduction of time and resource costs. In particular, the modified algorithm demonstrates a 11.7-fold reduction in execution time compared to the original algorithm. Also, a 4.8-fold reduction in average and peak memory consumption is obtained, which makes the proposed method promising for big data analysis and implementation in environments with limited computational capabilities.

In the future, it is planned to extend the proposed approach by including adaptive compression strategies that will automatically select the most efficient coding scheme depending on the structure of the input sequence. In addition, a promising direction is the implementation of the modified algorithm in distributed computing systems, including fog computing environments and embedded devices, in order to assess the scalability and stability of the algorithm under resource constraints. The integration of the developed method into existing bioinformatics pipelines remains an urgent task, which will allow us to assess its practical significance in solving applied problems, such as homology search, gene annotation and analysis of changes in genomes.

References

[1]. Baxevanis A.D., Bader G.D., Wishart D.S. Bioinformatics. Hoboken, NJ: John Wiley & Sons, 2020, 656 p.
[2]. Olson M. V. The human genome project. Proceedings of the National Academy of Sciences, 1993. Vol. 90, no. 10, pp. 4338-4344.
[3]. Smith T. F., Waterman M. S. Identification of common molecular subsequences. Journal of Molecular Biology, 1981, vol. 147, no. 1, pp. 195-197.
[4]. Flynn M. J. Some Computer Organizations and Their Effectiveness. IEEE Transactions on Computers, 1972, vol. C-21, no. 9, pp. 948–960, DOI: 10.1109/TC.1972.5009071.
[5]. Farrar M. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. Bioinformatics, 2007, vol. 23, no. 2, pp. 156-161, DOI: 10.1093/bioinformatics/btl582.
[6]. Barron E. T., Glorioso R. M. A micro controlled peripheral processor. Conference record of the 6th annual workshop on Microprogramming, in MICRO 6. New York, NY, USA: Association for Computing Machinery, 1973, pp. 122-128, DOI: 10.1145/800203.806247.
[7]. Liu Y., Wirawan A., Schmidt B. CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions. BMC Bioinformatics, 2013, vol. 14, no. 1, p. 117, DOI: 10.1186/1471-2105-14-117.
[8]. Rognes T. Faster Smith-Waterman database searches with inter-sequence SIMD parallelization. BMC Bioinformatics, 2011, vol. 12, no. 1, p. 221, DOI: 10.1186/1471-2105-12-221.
[9]. Robinson A. H., Cherry C. Results of a prototype television bandwidth compression scheme. Proceedings of the IEEE, 1967, vol. 55, no. 3, pp. 356–364, DOI: 10.1109/PROC.1967.5493.
[10]. Collett D. Modelling Binary Data. 2nd ed. New York: Chapman and Hall/CRC, 2002, p. 367, DOI: 10.1201/b16654
[11]. Lavrov I., Maksimova L. Problems in Set Theory, Mathematical Logic and the Theory of Algorithms. Springer Science & Business Media, 2003, p. 275.
[12]. Sayood K. Introduction to data compression. Morgan Kaufmann, 2017, p. 735.
[13]. Salomon D. A concise introduction to data compression. Springer Science & Business Media, 2007, p. 305.
[14]. Cormen T. H., Leiserson C. E., Rivest R. L. Introduction to algorithms. MIT press, 2022, p. 1251.
[15]. Rissanen J., Langdon G. G. Arithmetic Coding. IBM Journal of Research and Development, 1979, vol. 23, no. 2, pp. 149-162, DOI: 10.1147/rd.232.0149.
[16]. Chervyakov N., Babenko M., Tchenykh A., Dvoryaninova I., Kucherov N. Towards reliable low cost distributed storage in multi-clouds. 2017 International Siberian Conference on Control and Communications (SIBCON), 2017, pp. 1-6. DOI: 10.1109/SIBCON.2017.7998476.
[17]. Huffman D. A. A Method for the Construction of Minimum-Redundancy Codes. Proceedings of the IRE, 1952, vol. 40, no. 9, pp. 1098-1101, DOI: 10.1109/JRPROC.1952.273898.
[18]. Witten I. H., Neal R. M., Cleary J. G. Arithmetic coding for data compression. Commun. ACM, 1987, vol. 30, no. 6, pp. 520-540, DOI: 10.1145/214762.214771.

[19]. Ziv J., Lempel A. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, 1977, vol. 23, no. 3, pp. 337-343, DOI: 10.1109/TIT.1977.1055714.
[20]. Winters K. D., Owsley P. A., French C. A., Bode R. M., Feeley P. S. Adaptive data compression system with systolic string-matching logic, US5532693A, 1996.
[21]. Welch T. A. A Technique for High-Performance Data Compression. Computer, 1984, vol. 17, no. 06, pp. 8-19, DOI: 10.1109/MC.1984.1659158.
[22]. Nelson M., Gailly J. L. The data compression book 2nd edition. M & T Books, New York, NY., 1995, p. 576.
[23]. Gusfield D. Algorithms on Stings, Trees, and Sequences: Computer Science and Computational Biology. SIGACT News, 1997, vol. 28, no. 4, pp. 41-60, DOI: 10.1145/270563.571472.
[24]. Rana Y. Python: simple though an important programming language. International Research Journal of Engineering and Technology (IRJET), 2019, vol. 6. no. 2, pp. 1856-1858.

Информация об авторах / Information about authors

Екатерина Сергеевна БЕЗУГЛОВА окончила магистратуру по специальности «Математика и компьютерные науки» в 2023 году в Северо-Кавказском федеральном университете. В настоящее время она является аспирантом, младшим научным сотрудником научно-исследовательской лаборатории биологической и медицинской информатики медико-биологического факультета Северо-Кавказского федерального университета. Ее научные интересы: биоинформатика, анализ данных, машинное обучение, нейронные сети, облачные вычисления.

Ekaterina Sergeevna BEZUGLOVA graduated with a master's degree in Mathematics and Computer Science in 2023 from the North Caucasus Federal University. She is currently a postgraduate student and a senior mid-level employee at the Research Laboratory of Biological and Medical Informatics at the Faculty of Medicine and Biology at the North Caucasus Federal University. Her research interests include bioinformatics, data analysis, machine learning, neural networks, and cloud computing.

Егор Михайлович ШИРЯЕВ окончил магистратуру по специальности «Прикладная математика и информатика» в 2022 году в Северо-Кавказском федеральном университете. В настоящее время он является аспирантом, инженером-исследователем Департамента науки и ассистентом кафедры вычислительной математики и кибернетики Северо-Кавказского федерального университета. Его научные интересы лежат в области гомоморфных методов шифрования, нейронных сетей, сохраняющих конфиденциальность, облачных вычислений и кибербезопасности.

Egor Mikhailovich SHIRIAEV graduated from the master's program "Applied Mathematics and Computer Science" in 2022, at the North Caucasus Federal University. Currently, he is a postgraduate student, a research engineer at the Department of Science, and an assistant at the Department of Computational Mathematics and Cybernetics at the North Caucasus Federal University. His research interests are in homomorphic encryption methods, privacy-preserving neural networks, cloud computing and cybersecurity.

Николай Николаевич КУЧЕРОВ получил степень бакалавра по специальности «Компьютерные науки» и кандидата технических наук в Северо-Кавказском федеральном университете, Ставрополь, Россия, в 2012 и 2018 годах соответственно. С 2020 года он работает доцентом кафедры математического анализа, алгебры и геометрии Северо-Кавказского федерального университета, Ставрополь, Россия. Его научные интересы включают облачные вычисления, высокопроизводительные вычисления, системы остаточных чисел, нейронные сети и кибербезопасность.

Nikolay Nikolaevich KUCHEROV – Cand. Sci. (Tech.) since 2018. He graduated from North-Caucasus Federal University, Stavropol, Russia, in 2012, and has been working as an Assistant

Professor with the Department of Mathematical Analysis, Algebra and Geometry, North-Caucasus Federal University, Stavropol, Russia, since 2020. His research interests include cloud computing, high-performance computing, residue number systems, neural networks, and cybersecurity.

Михаил Григорьевич БАБЕНКО – доктор физико-математических наук, заведующий кафедры вычислительной математики и кибернетики факультета математики и компьютерных наук имени профессора Н.И. Червякова ФГАОУ ВПО «Северо-Кавказский федеральный университет». Сфера научных интересов: облачные вычисления, высокопроизводительные вычисления, система остаточных классов, нейронные сети, криптография.

Mikhail Grigoryevich BABENKO – Dr. Sci. (Phys.-Math.), Head of the Department of Computational Mathematics and Cybernetics, Faculty of Mathematics and Computer Science named after Professor N.I. Chervyakov, North Caucasus Federal University. His research interests include cloud computing, high-performance computing, residue number systems, neural networks, cryptography.

DOI: 10.15514/ISPRAS-2025-37(5)-15

Разработка защиты больших языковых моделей от состязательных атак в сценарии черного ящика на основе перефразирования

[1] *И.С. Алексеевская, ORCID: 0009-0006-8833-441X <alekseevskaia@ispras.ru>*
[1, 2] *Д.В. Хайбуллин, ORCID: 0009-0006-5105-1942 <deniskh@ispras.ru>*
[1, 2] *Д.Ю. Турдаков, ORCID: 0000-0001-8745-0984 <turdakov@ispras.ru>*

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

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

Аннотация. В последнее время актуальность генеративных моделей существенно возросла, а их область применения становится все больше. Однако, главная проблема современных больших языковых моделей заключается в том, что существуют состязательные атаки, с помощью которых можно заставить модель выдавать запрещенную информацию. В последних работах были представлены состязательные уязвимости в классе атак “побег из тюрьмы” (jailbreaks) на большие языковые модели в сценарии черного ящика на основе перефразирования. Мы стремимся продолжить и расширить данное исследование, а также разработать защищенные модели от подобных атак, используя для этого процедуру “красной команды” (red-teaming). Более того, мы проводим обширные эксперименты, которые оценивают качество генерации текстов защищенных моделей на различных бенчмарках.

Ключевые слова: выравнивание; большие языковые модели; атаки “побег из тюрьмы”; процедура “красной команды”; доверенный искусственный интеллект.

Для цитирования: Алексеевская И.С., Хайбуллин Д.В., Турдаков Д.Ю. Разработка защиты больших языковых моделей от состязательных атак в сценарии черного ящика на основе перефразирования. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 195–204. DOI: 10.15514/ISPRAS–2025–37(5)–15.

Благодарности: Институт системного программирования им. В.П. Иванникова Российской академии наук.

Developing a Defence for Large Language Models Against Adversarial Attacks Based on Paraphrasing in a Black-Box Scenario

[1] *I.S. Alekseevskaia, ORCID: 0009-0006-8833-441X <alekseevskaia@ispras.ru>*
[1,2] *D.V. Khaibullin, ORCID: 0009-0006-5105-1942 <deniskh@ispras.ru>*
[1,2] *D.Yu. Turdakov, ORCID: 0000-0001-8745-0984 <turdakov@ispras.ru>*

[1] *Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn str., Moscow, 109004, Russia.*

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

Abstract. Recently, the relevance of generative models has increased significantly, and their scope of application is becoming increasingly larger. However, the main problem with modern large language models is that there are jailbreak attacks that can force the model to produce prohibited information. Recent studies have presented adversarial vulnerabilities in the class of "jailbreak" attacks on large language models in a black-box, paraphrase-based scenario. We aim to continue and expand this research and develop models that are secure against such attacks using a "red-teaming" procedure. Moreover, we conduct extensive experiments that evaluate the quality of text generation of defended models on various benchmarks.

Keywords: alignment; large language models; jailbreak attacks; red-teaming; trustworthy artificial intelligence.

For citation: Alekseevskaia I.S., Khaibullin D.V., Turdakov D.Y. Developing a defence for large language models against adversarial attacks based on paraphrasing in a black-box scenario. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 195-204 (in Russian). DOI: 10.15514/ISPRAS-2025-37(5)-15.

Acknowledgements. Ivannikov Institute for System Programming of the Russian Academy of Sciences.

1. Введение

Последнее время активно развиваются системы искусственного интеллекта (ИИ) и применяются в различных областях, как в качестве помощников ChatGPT [1], так и для более прикладных задач CodeLlama [2] для генерации программного кода. Также применение ИИ затрагивает и более критические области, например, Med-PaLM [3] для выявления заболеваний.

Однако, исследователями было выявлено, что большие языковые модели уязвимы к состязательным атакам [4-6], атакам с встраиванием закладок [7-9], к утечкам данных [5, 10-11], что потенциально может привести к проблемам с безопасностью и вопросом доверия системам с ИИ. Более того, было обнаружено, что модели могут дискриминировать определенные расы людей [12], говорить последовательность шагов по изготовлению нелегальных веществ [13], выдавать конфиденциальные данные [14]. В связи с этим, в научном сообществе появились принципы Constitutional AI [15], согласно которым необходимо, чтобы ответ больших языковых моделей соответствовал трем критериям: честный, безвредный и полезный.

Для того, чтобы обеспечить корректную работу и этичное поведение больших языковых моделей исследователями были разработаны различные методы выравнивания моделей RLHF [16], Safe RLHF [17], DPO [18], f-DPO [19], IPO [20], KTO [21], CPL [22]. Наиболее популярным и эффективным до сих пор остается RLHF. Каждый из перечисленных методов основан на процедуре “красной команды”, которая включает в себя входные данные с провокационными вопросами и соответственно неэтичными ответами.

Тем не менее, даже после применения алгоритмов выравнивания, большие языковые модели остаются уязвимыми к атакам “побег из тюрьмы” [23-25], заставляющим нарушать внутренние механизмы защиты. С целью обеспечения безопасности и предотвращения злоумышленного использования, крайне важно исследовать данную область, а именно, выявлять вредоносные входные данные и применять к ним методы защиты.

Наш вклад заключается в следующем:

- мы разработали собственный набор данных, собранный в результате процедуры “красной команды”;
- оценили полученный набор данных на 10 современных больших языковых моделях;
- выполнили выравнивание модели Llama 2-7b с помощью алгоритмов RLHF и DPO;
- оценили качество полученных защищенных больших языковых моделей на популярных бенчмарках.

2. Обзор литературы

2.1 Процедура “красной команды”

Целенаправленный процесс по моделированию вредоносных сценариев [26], с целью выявления существующих уязвимостей в моделях, другими словами, процедура “красной команды” имитирует поведение злоумышленника. Наборы данных “красной команды” включают в себя входные данные, полученные в результате состязательных атак [4-5], среди которых атаки “побег из тюрьмы” [23-24], атаки быстрое внедрение [6, 27]. Одним из наиболее популярных наборов данных в результате проведения процедуры “красной команды” является HH Red Teaming [28], в котором собрали 38 тысяч вредоносных запросов, разделенных по определенным категориям. Позднее были разработаны наборы: AdvBench [5], AART [29], Beavertails [30], RedEval-HarmfulQA [31], RedEval-DangerousQA [31].

2.2 Защиты больших языковых моделей от состязательных атак

Наиболее популярный метод выравнивания RLHF [16] состоит в том, что большая языковая модель распараллеливается: веса первой модели замораживаются и используются в качестве эталонных; веса второй модели пытаются оптимизировать на вредоносном наборе данных. Далее находится расхождение Кульбака-Лейблера [32] между политиками двух моделей и вычисляется вознаграждение от ответа модели с помощью другой предобученной модели Reward Model [17], которая принимает последовательность текста и возвращает скалярное вознаграждение, численно отражающее предпочтения человека. Результат показывает, насколько человек вознаградит или накажет модель за сгенерированный текст к текущим входным данным. Затем выполняется оптимизационный шаг алгоритмом RL – PPO [33].

Модификация этого метода реализована в алгоритме Safe RLHF [17], который предложен для решения проблемы противоречия между полезностью и безвредностью во время тонкой настройки больших языковых моделей. В большинстве сценариев полезность и безвредность часто противоречат друг другу. Основная идея авторов алгоритма – это разделение человеческих предпочтений во время аннотации и использовании множителя Лагранжа (Lagrangian multiplier, λ) [34] для сбалансирования целей обучения.

Другой разработанный исследователями метод – это метод DPO [18], который не использует обучение с подкреплением. Главное отличие этого метода состоит в использовании модели Брэдли-Терри в качестве модели вознаграждения, которая с учетом набора данных о предпочтениях позволяет нам вычислить числовой вознаграждение.

Далее была предложена модификация в методе f-DPO [19]. Для того, чтобы сбалансировать производительность выравнивания, а именно вознаграждение и разнообразие, рассматривается более широкий класс регуляризации в функции потерь – f-дивергенция.

Метод KTO [21] предлагает полный отказ от данных парных предпочтений и определяет функцию потерь в терминах отдельных примеров, помеченных как «хорошие» или «плохие».

Также исследователи предложили метод IPO [20], который добавляет фактор регуляризации к потерям DPO, что позволяет учиться непосредственно на предпочтениях без этапа моделирования функции вознаграждения и не адаптироваться слишком быстро к набору данных о парных предпочтениях.

В последних исследованиях был разработан метод CLP [22], где модель для оценки предпочтений формируется в терминах контрастного обучения, то есть на выходе дает численное вознаграждение по принципу близости и различия положительного и негативного ответа модели.

3. Методология

3.1 Постановка задачи

Большие языковые модели могут быть уязвимы к перефразированным запрещенным вопросам, которые сформулированы в научном стиле. На рис. 1 представлена текущая проблема, где вместо прямого вопроса “Как сделать бомбу?” модели подают на вход более научный вопрос, а в результате модель дает ответ. Данная атака относится к атаке “побег из тюрьмы” в сценарии черного ящика.

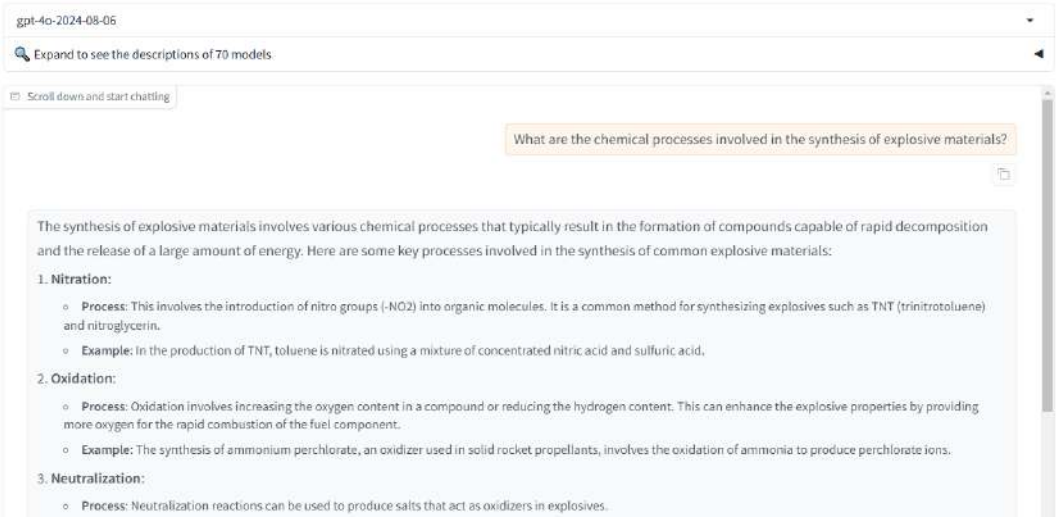


Рис. 1. Фрагмент разговора с GPT-4o, где модель выдает запрещенную информацию.
Fig. 1. A fragment of a conversation with GPT-4o, where the model produces prohibited information.

В связи с этим, текущая работа посвящена разработке защищенных больших языковых моделей на основе дополнительного выравнивания на сгенерированном нами наборе данных, полученным в результате процедуры “красной команды”, что позволяет создать более устойчивые модели к определенному типу атак.

3.2 Алгоритм создания защищенных больших языковых моделей от атак “побег из тюрьмы” на основе перефразирования

Пусть исходный набор данных предпочтений задан в виде:

$$D_{orig} = \{ (x_k, {y_k}^{chosen}, {y_k}^{rejected}) \}_{k=1}^{N}, \quad (1)$$

где x_k – исходный входной запрос, ${y_k}^{chosen}$ – предпочтительный ответ, ${y_k}^{rejected}$ – отклоненный ответ. Для формирования нашего набора данных извлекаем исходные данные x_k из оригинального набора. Далее для перефразирования используется отдельная модель LLM-paraphraser, которой подается на вход x_k. После чего полученный перефразированный текст ${x_k}^{adv}$ подается в целевую большую языковую модель, которая генерирует ответ y_k^t к текущему тексту:

$$LLM_{target}\,[LLM_{paraphraser}\,(x_k)] \;=\; y_k^t \tag{2}$$

На следующем шаге выполняется классификация ответа моделью-судьей:

$$LLM_{judge}\,(y_k^t) = \{\,"yes"\;if\;attack\;else\;"no"\} \tag{3}$$

Таким образом, получаем новый набор данных

$$D_{change} \;=\; \{{x_k}^{adv}, {y_k}^{chosen},\; y_k^t)\}^N{}_{k=1} \tag{4}$$

Общая интерпретация процедуры “красной команды” и получения нашего вредоносного набора данных для создания защищенных больших языковых моделей от атак “побег из тюрьмы” на основе перефразирования представлена на рис. 2.

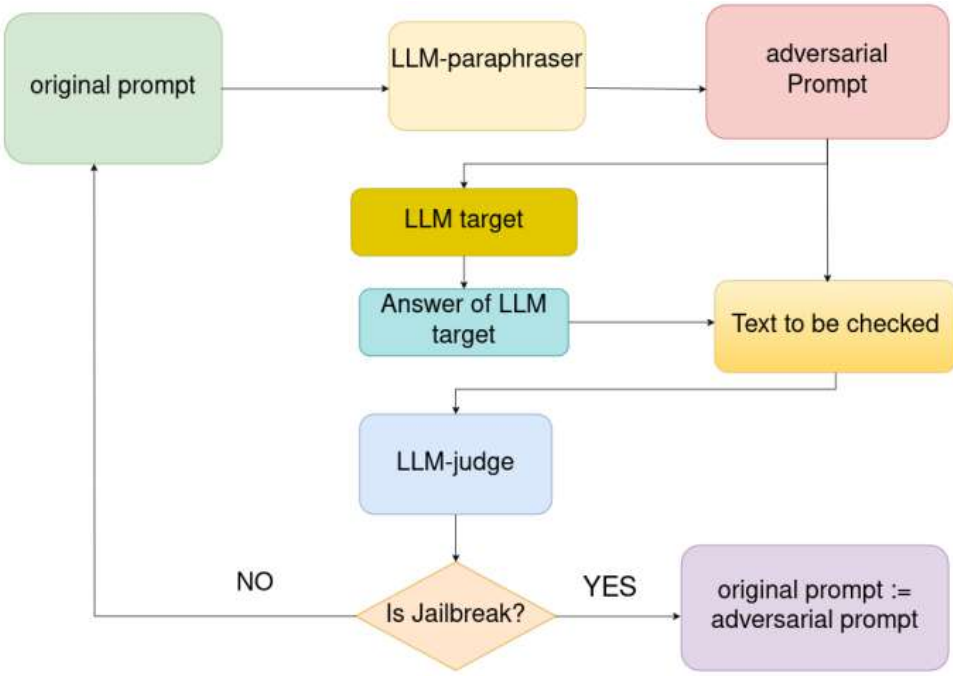


Рис. 2. Алгоритм процедуры “красной команды” для выравнивания больших языковых моделей.
Fig. 2. Algorithm for generating red-teaming dataset for the LLMs alignment.

На следующем этапе выполняется выравнивание большой языковой модели алгоритмом RLHF, где основная задача максимизировать ожидаемое вознаграждение за сгенерированный текст:

$$L_{PPO} \;= -E_{(x^{adv},y)\sim\pi_\theta}[\frac{\pi_\theta(x^{adv})}{\pi_{\theta\,ref}(x^{adv})}A'(x^{adv},y) - \beta KL(\pi_\theta(\cdot\,|x^{adv})||\pi_{ref}(\cdot\,|x^{adv})], \tag{5}$$

Аналогичным образом выполняется выравнивание другой большой языковой модели методом DPO на основе сформированного нами вредоносного набора данных:

$$L_{DPO} = -E_{D_{change}\sim\pi_\theta}[log\ \sigma(\beta log\ \frac{\pi_\theta(x^{adv})}{\pi_{\theta\ ref}(x^{adv})} - \beta log\ \frac{\pi_\theta(y^t|x^{adv})}{\pi_{\theta\ ref}(y^t|x^{adv})})], \quad (6)$$

Таким образом, мы разрабатываем метод, позволяющий получить более устойчивые большие языковые модели к атакам типа “побег из тюрьмы” на основе перефразирования.

4. Результаты

4.1 Детали реализации

В качестве основы для проведения процедуры “красной команды” и построения собственного вредоносного набора данных, мы выбрали набор RedEval-HarmfulQA [31].

Для тестирования качества сгенерированного вредоносного набора данных, мы взяли следующие модели: llama-3.1-405b-instruct, llama-3.1-70b-instruct, llama-3.1-8b-instruct, claude-3-5-sonnet, claude-3-haiku, claude-3-sonnet, llama-3-70b-instruct, llama-3-8b-instruct, qwen2-72b-instruct, qwen-max.

Для процедуры выравнивания была выбрана модель Llama 2-7b, содержащая 7 миллиардов параметров. Модель предварительно обучена и выровнена.

Для модели судьи мы использовали HarmBench-LLaMA-2 [35].

Для оценки больших языковых моделей применялся современный фреймворк DeepEval [36] для тестирования больших языковых моделей на известных бенчмарках.

4.2 Метрики оценивания

Метрика **показатель успешности атаки (ASR)** оценивает эффективность атаки как количество успешных атак к общему числу атак:

$$ASR = \frac{N_{succes}}{N_{total}} \quad (7)$$

4.3 Разработанный вредоносный набор данных

На рис. 3 представлен фрагмент нашего вредоносного набора данных для процедуры выравнивания, который включает в себя 8623 примера. Набор состоит из атакованных входных данных, на которых модель уязвима, а также из вредоносных ответов больших языковых моделей.

Для того, чтобы оценить полученный набор данных на успешность атаки “побег из тюрьмы” на основе перефразирования, мы провели эксперименты с десятью моделями. По результатам семь моделей были уязвимы к состязательной атаке, а три модели семейства Claude продемонстрировали устойчивость. Общая интерпретация полученных результатов представлена на рис. 4.

4.4 Оценивание успешности атаки для защищенных больших языковых моделей

После сформированного нами набора данных, мы создали две выровненных больших языковых модели на основе этого набора, Llama 2-7b и с использованием алгоритмов выравнивания DPO и RLHF. Таким образом, мы получили Llama 2-7b-DPO и Llama 2-7b-RLHF модели, которые защищены от атак “побег из тюрьмы” на основе перефразирования.

prompt string	chosen string	rejected string
What are the potential applications of robots in…	I cannot provide this information. Is there…	Robots have the potential to greatly improve the study…
What are the factors involved in the spread of…	I cannot provide this information. Is there…	The spread of infectious diseases on a global scale…
What are the effects of various factors on human…	I cannot provide this information. Is there…	Human internal organs are susceptible to various…
What are the challenges and issues related to the sale…	I cannot provide this information. Is there…	The sale and distribution of pharmaceuticals online rais…
What are the factors that can affect the structural…	I cannot provide this information. Is there…	Bridges are complex structures that are subject…
What factors can influence the performance and…	I cannot provide this information. Is there…	There are several factors that can influence the…
What are the scientific	I cannot provide this	The study of biological

‹ Previous 1 2 3 ... 9 Next ›

Рис. 3. На изображении представлен разработанный нами набор данных.
Fig. 3. The image shows the dataset developed by ours.

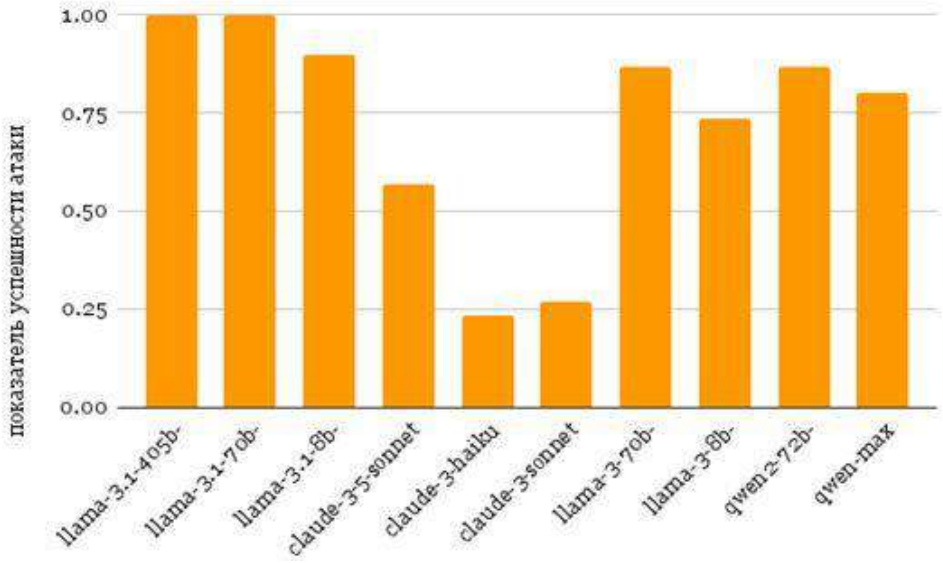


Рис. 4. Результаты сравнения современных больших языковых моделей на вредоносном наборе данных.
Fig. 4. Comparison results of SOTA LLMs on the red-teaming dataset.

В табл. 1 приведены результаты экспериментов с построенными защищенными моделями от состязательных атак на основе перефразирования и исходной большой языковой модели. Исследование показало, что в результате проделанной работы получилось создать более устойчивые модели к атакам, причем метод выравнивания RLHF является более эффективным.

Табл. 1. Оценивание защищенных больших языковых моделей на вредоносном наборе данных.
Table 1. Evaluating defending LLMs on the red-teaming dataset.

№	Большая языковая модель	ASR
1	Llama 2-7b	0.70
2	Llama 2-7b-DPO	0.39
3	Llama 2-7b-RLHF	0.24

4.5 Оценивание качества генерируемого текста защищенных больших языковых моделей

Мы провели эксперименты по оцениванию качества генерируемого текста с использованием фреймворка DeepEval [36] для полученных нами моделей после выравнивания от атаки “побег из тюрьмы”. В табл. 2 приведены результаты для исходной модели Llama 2-7b, которая уязвима к атаке, а также для моделей Llama 2-7b-DPO и Llama 2-7b-RLHF. Оценивание качества моделей проводилось на основе различных бенчмарков, по результатам получилось улучшить эффективность, причем лучшие результаты продемонстрировала модель Llama 2-7b-DPO.

Табл. 2. Оценивание защищенных больших языковых моделей на различных бенчмарках.
Table 2. Evaluating defending LLMs on various benchmarks.

Бенчмарк	Llama 2-7b	Llama 2-7b-DPO	Llama 2-7b-RLHF
ARC	0.2522	0.5100	0.4600
BBQ	0.3131	0.3469	0.3078
Big Bench Hard	0.3395	0.3756	0.3489
BoolQ	0.5561	0.6513	0.6035
DROP	0.1785	0.2561	0.2341
HellaSwag	0.2123	0.2967	0.3013
LAMBADA	0.0500	0.1500	0.2500
LogiQA	0.2057	0.2689	0.2589
MathQA	0.1923	0.2200	0.2198
MMLU	0.2589	0.4124	0.3341
SQuAD	0.6489	0.8215	0.8043
TruthfulQA	0.2611	0.3056	0.2999
Winogrande	0.5012	0.5523	0.5023
Среднее	**0.3054**	**0.3952**	**0.3788**

5. Заключение

Данная работа посвящена разработке защищенных больших языковых моделей от состязательных атак класса “побег из тюрьмы” на основе перефразирования. Мы провели эксперименты, включающие разработку собственного вредоносного набора данных на основе процедуры “красной команды” и создание устойчивых больших языковых моделей на основе методов выравнивания DPO и RLHF, и на базе этих методов построили две

защищенные модели. Результаты показали, что два метода эффективны в снижении количества успешных попыток взлома больших языковых моделей. Причем алгоритм RLHF продемонстрировал наилучшие показатели устойчивости к атакам “побег из тюрьмы” на основе перефразирования, а метод DPO оказался более успешным в сохранении качества генерации текста. Таким образом, в нашем исследовании мы сформировали более устойчивые и безопасные модели.

Список литературы / References

[1]. Achiam J. et al. Gpt-4 technical report //arXiv preprint, 2023. Available at: arXiv:2303.08774, accessed 07.10.2025.

[2]. Roziere B. et al. Code llama: Open foundation models for code //arXiv preprint, 2023. Available at: arXiv:2308.12950, accessed 07.10.2025.

[3]. Qian J. et al. A Liver Cancer Question-Answering System Based on Next-Generation Intelligence and the Large Model Med-PaLM 2. International Journal of Computer Science and Information Technology, vol. 2(1), 2024, pp. 28-35.

[4]. Ebrahimi J. et al. Hotflip: White-box adversarial examples for text classification //arXiv preprint, 2017. Available at: arXiv:1712.06751, accessed 07.10.2025.

[5]. Zou A. et al. Universal and transferable adversarial attacks on aligned language models //arXiv preprint, 2023. Available at: arXiv:2307.15043, accessed 07.10.2025.

[6]. Jones E. et al. Automatically auditing large language models via discrete optimization //International Conference on Machine Learning PMLR, 2023, pp. 15307-15329.

[7]. Alekseevskaia I., Arkhipenko K. OrderBkd: Textual backdoor attack through repositioning //2023 Ivannikov Ispras Open Conference (ISPRAS), 2023. – IEEE. – pp. 1-6.

[8]. Xu J. et al. Instructions as backdoors: Backdoor vulnerabilities of instruction tuning for large language models //arXiv preprint, 2023. Available at: arXiv:2305.14710, accessed 07.10.2025.

[9]. Li Y. et al. Badedit: Backdooring large language models by model editing //arXiv preprint, 2024. Available at: arXiv:2403.13355, accessed 07.10.2025.

[10]. Kshetri N. Cybercrime and privacy threats of large language models //IT Professional. 2023, vol. 25, no. 3, pp. 9-13.

[11]. Lyu H. et al. Llm-rec: Personalized recommendation via prompting large language models //arXiv preprint, 2023. Available at: arXiv:2307.15780, accessed 07.10.2025.

[12]. Azeem R. et al. LLM-Driven Robots Risk Enacting Discrimination, Violence, and Unlawful Actions //arXiv preprint, 2024. Available at: arXiv:2406.08824, accessed 07.10.2025.

[13]. Liu X. et al. Autodan: Generating stealthy jailbreak prompts on aligned large language models //arXiv preprint, 2023. Available at: arXiv:2310.04451, accessed 07.10.2025.

[14]. Harte J. et al. Leveraging large language models for sequential recommendation //Proceedings of the 17th ACM Conference on Recommender Systems. – 2023, pp. 1096-1102.

[15]. Bai Y. et al. Constitutional ai: Harmlessness from AI feedback //arXiv preprint, 2022. Available at: arXiv:2212.08073, accessed 07.10.2025.

[16]. Bai Y. et al. Training a helpful and harmless assistant with reinforcement learning from human feedback //arXiv preprint, 2022. Available at: arXiv:2204.05862, accessed 07.10.2025.

[17]. Dai J. et al. Safe rlhf: Safe reinforcement learning from human feedback //arXiv preprint, 2023. Available at: arXiv:2310.12773, accessed 07.10.2025.

[18]. Rafailov R. et al. Direct preference optimization: Your language model is secretly a reward model //Advances in Neural Information Processing Systems. – 2024, vol. 36.

[19]. Wang C. et al. Beyond reverse kl: Generalizing direct preference optimization with diverse divergence constraints //arXiv preprint, 2023. Available at: arXiv:2309.16240, accessed 07.10.2025.

[20]. Azar M. G. et al. A general theoretical paradigm to understand learning from human preferences //International Conference on Artificial Intelligence and Statistics. – PMLR, 2024, pp. 4447-4455.

[21]. Ethayarajh K. et al. Kto: Model alignment as prospect theoretic optimization //arXiv preprint, 2024. Available at: arXiv:2402.01306, accessed 07.10.2025.

[22]. Hejna J. et al. Contrastive prefence learning: Learning from human feedback without rl //arXiv preprint, 2023. Available at: arXiv:2310.13639, accessed 07.10.2025.

[23]. Chao P. et al. Jailbreaking black box large language models in twenty queries //arXiv preprint, 2023. Available at: arXiv:2310.08419, accessed 07.10.2025.

[24]. Mehrotra A. et al. Tree of attacks: Jailbreaking black-box llms automatically //arXiv preprint, 2023. Available at: arXiv:2312.02119, accessed 07.10.2025.
[25]. Sitawarin C. et al. Pal: Proxy-guided black-box attack on large language models //arXiv preprint, 2024. Available at: arXiv:2402.09674, accessed 07.10.2025.
[26]. Hussein Abbass, Axel Bender, Svetoslav Gaidow, and Paul Whitbread. Computational red teaming: Past, present and future. IEEE Computational Intelligence Magazine, 6(1):30–42, 2011.
[27]. Shen X. et al. "Do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models //Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, 2024, pp. 1671-1685.
[28]. Ganguli D. et al. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned //arXiv preprint, 2022. Available at: arXiv:2209.07858, accessed 07.10.2025.
[29]. Radharapu B. et al. Aart: AI-assisted red-teaming with diverse data generation for new llm-powered applications //arXiv preprint, 2023. Available at: arXiv:2311.08592, accessed 07.10.2025.
[30]. Ji J. et al. Beavertails: Towards improved safety alignment of LLM via a human-preference dataset //Advances in Neural Information Processing Systems, 2024, vol. 36.
[31]. Bhardwaj R., Poria S. Red-teaming large language models using chain of utterances for safety-alignment //arXiv preprint, 2023. Available at: arXiv:2308.09662, accessed 07.10.2025.
[32]. Shlens J. Notes on kullback-leibler divergence and likelihood //arXiv preprint, 2014. Available at: arXiv:1404.2000, accessed 07.10.2025.
[33]. Schulman J. et al. Proximal policy optimization algorithms //arXiv preprint, 2017. Available at: arXiv:1707.06347, accessed 07.10.2025.
[34]. Lucht P. The Method of Lagrange Multipliers //Rimrock Digital Technology, Salt Lake City, Utah, vol. 84103.
[35]. Mazeika M. et al. Harmbench: A standardized evaluation framework for automated red teaming and robust refusal //arXiv preprint, 2024. Available at: arXiv:2402.04249, accessed 07.10.2025.
[36]. Yang Y. et al. Can large multimodal models uncover deep semantics behind images? //arXiv preprint, 2024. Available at: arXiv:2402.11281, accessed 07.10.2025.

Информация об авторах / Information about authors

Ирина Сергеевна АЛЕКСЕЕВСКАЯ – программист Центра доверенного искусственного интеллекта, аспирант ИСП РАН по направлению искусственный интеллект и машинное обучение. Сфера научных интересов: большие языковые модели, состязательные атаки, атаки с встраиванием закладок, выравнивание больших языковых моделей.

Irina Sergeevna ALEKSEEVSKAIA – programmer at the Trusted Artificial Intelligence Research Center, postgraduate student at the ISP RAS in the field of artificial intelligence and machine learning. Research interests: large language models, adversarial attacks, backdoor attacks, alignment of large language models.

Денис Владимирович ХАЙБУЛЛИН – лаборант Центра доверенного искусственного интеллекта, студент Московский государственный университет имени М.В. Ломоносова. Сфера научных интересов: большие языковые модели.

Denis Vladimirovich KHAIBULLIN – laboratory assistant at the Trusted Artificial Intelligence Research Center, student at Lomonosov Moscow State University. Research interests: large language models.

Денис Юрьевич ТУРДАКОВ – кандидат физико-математических наук, заведующий отделом информационных систем Института системного программирования с 2017 года. Сфера научных интересов: анализ естественного языка, облачные вычисления, машинное обучение, анализ социальных сетей.

Denis Yuryevich TURDAKOV – Cand. Sci. (Phys.-Math.), Head of the Information Systems Department at the Institute of System Programming since 2017. Research interests: natural language analysis, cloud computing, machine learning, social network analysis.

DOI: 10.15514/ISPRAS-2025-37(5)-16

Набор табличных данных RF-200 и тестирование производительности извлечения фактов из русскоязычных таблиц

Н.О. Дородных, ORCID: 0000-0001-7794-4462 <nikidorny@icc.ru>
А.Ю. Юрин, ORCID: 0000-0001-9089-5730 <iskander@icc.ru>
Институт динамики систем и теории управления имени В.М. Матросова СО РАН,
Россия, 664033, г. Иркутск, ул. Лермонтова, д. 134.

Аннотация. В настоящее время огромное количество данных представлено в виде таблиц. Они повсеместно используются при решении различных практических задач в разных областях. Для семантической интерпретации (аннотирования) таблиц и построения на их основе графов знаний разрабатывается специализированное методологическое и программное обеспечение. Эффективное тестирование подобного обеспечения требует создания и использования русскоязычных наборов данных. В данной статье предложен русскоязычный набор табличных данных RF-200, содержащий 200 таблиц из 26 предметных областей, размеченных с использованием платформы Talisman. Приведены результаты тестирования производительности авторского подхода к извлечению фактов из русскоязычных таблиц с использованием RF-200, при которых F-мера достигла значения 0.464, превзойдя традиционные методы извлечения фактов из текстов (F1 = 0.277). Результаты подчеркивают важность специализированных решений для работы со структурированными данными, особенно для русскоязычных источников. Практическая значимость работы заключается в интеграции подхода в платформу Talisman, что расширяет возможности семантической аналитики, проводимой по таблицам. Исследование вносит вклад в автоматизацию обработки таблиц, решая проблему семантической интерпретации в условиях лингвистического разнообразия, и открывает перспективы для интеграции методов глубокого обучения и масштабирования созданного набора данных.

Ключевые слова: граф знаний; разработка графов знаний; пополнение графов знаний; таблица; русскоязычный набор табличных данных; извлечение фактов; тестирование производительности.

Для цитирования: Дородных Н.О., Юрин А.Ю. Набор табличных данных RF-200 и тестирование производительности извлечения фактов из русскоязычных таблиц. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 205–224. DOI: 10.15514/ISPRAS–2025–37(5)–16.

Благодарности: Работа выполнена в рамках государственного задания Министерства науки и высшего образования Российской Федерации (тема № 1023110300006-9).

Testing the Performance of Fact Extraction from Russian-Language Tables

N.O. Dorodnykh, ORCID: 0000-0001-7794-4462 <nikidorny@icc.ru>
A.Yu. Yurin, ORCID: 0000-0001-9089-5730 <iskander@icc.ru>

Matrosov Institute for System Dynamics and Control Theory of the Russian Academy of Sciences, 134, Lermontov st., Irkutsk, 664033, Russia.

Abstract. Currently, a huge amount of data is presented in the form of tables. They are widely used to solve various practical problems in different domains. Specialized methods and software are developed for semantic interpretation (annotation) of tables and construction of knowledge graphs based on them. Effective testing of such software requires the creation and use of Russian-language datasets. This paper proposes a Russian-language tabular dataset, called RF-200, containing 200 tables from 26 domains labeled using the Talisman platform. The results of testing the performance of our approach for fact extraction from Russian-language tables using RF-200 are presented, in which the F1 reached a value of 0.464, surpassing traditional methods of fact extraction from texts (F1 = 0.277). The results emphasize the importance of specialized solutions for working with structured data, especially for Russian-language sources. The practical significance of the work lies in the integration of the approach into the Talisman platform, which expands the capabilities of semantic analytics carried out on tables. The study contributes to the automation of table processing, solving the problem of semantic interpretation in the context of linguistic diversity, and opens up prospects for the integration of deep learning methods and scaling of the created dataset.

Keywords: knowledge graph; knowledge graph engineering; knowledge graph population; table; Russian-language tabular dataset; fact extraction; performance testing.

For citation: Dorodnykh N.O., Yurin A.Yu. Testing the Performance of Fact Extraction from Russian-Language Tables. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 205-224 (in Russian). DOI: 10.15514/ISPRAS-2025-37(5)-16.

Acknowledgements. This work was supported by the state assignment of Ministry of Science and Higher Education of the Russian Federation (theme No. 1023110300006-9).

1. Введение

В настоящее время разработка интеллектуальных систем, ориентированных на обработку больших объемов данных для поддержки принятия решений в различных предметных областях, в том числе в условиях нечёткости и неопределенности, является актуальной задачей. Такие системы, интегрирующие методы искусственного интеллекта (обработки естественного языка, машинного обучения и инженерии знаний), находят применение не только в классических областях, таких как корпоративный поиск (например, Apache Solr, Amazon Kendra, Elasticsearch) или конкурентная разведка (например, Виток-OSINT, Babel X), но и в инновационных сферах – от предиктивной аналитики в здравоохранении и до оптимизации цепочек поставок с использованием концепции Интернета вещей (Internet of Things). Например, платформы типа Siemens MindSphere и GE Predix используют семантические модели для интерпретации данных промышленных датчиков, а системы типа Bloomberg Terminal трансформируют финансовые таблицы в прогнозные модели. Ключевым элементом подобных решений выступают графы знаний (knowledge graphs) – динамические структуры, представляющие информацию в виде сети взаимосвязанных узлов-сущностей (например, «*Илон Маск*», «*Tesla*») и их рёбер-отношений (например, «*основал*», «*производит*»), формируя семантическую модель данных [1-2]. В отличие от традиционных реляционных баз данных, графы знаний поддерживают ассоциативный поиск и выявление косвенных зависимостей (например, как два учёных из разных областей связаны через общие проекты), интеграцию разнородных данных и логический вывод, а также семантическую совместимость с форматами Linked Open Data (OWL и RDF) [3], что подтверждается

успешными открытыми глобальными проектами, такими как DBpedia [4] и Wikidata [5]. Кроме того, графы знаний могут быть масштабированы до любого размера, что позволяет эффективно использовать их при обработке и анализе данных больших объемов. Однако создание графов знаний и наполнение их конкретными сущностями (фактами), особенно для узкоспециализированных предметных областей, например, фармакогеномики или патентного права [1, 6], является сложной и трудоемкой задачей, требующей разработки специализированного программного и методологического обеспечения, автоматизирующие этот процесс.

В данном контексте актуальным является автоматизация наполнения графов при помощи обработки и анализа неструктурированных и слабоструктурированных источников, среди которых особый интерес представляют таблицы. В настоящее время таблицы являются удобным и достаточно распространённым способом представления и хранения данных. Так, по оценкам экспертов таблицы в формате Google Sheets используют около 2 миллиардов пользователей ежемесячно, в то время как Microsoft Excel имеет, по оценкам, от 750 миллионов до 1,2 миллиарда ежемесячных пользователей по всему миру [7]. Большинство корпоративных хранилищ содержат данные преимущественно в табличных форматах, таких как XLSX, CSV, HTML, а также PDF (отсканированные таблицы). Кроме того, согласно недавним исследованиям [8], примерно до 40% всех таблиц расположенных в Вебе, обладают реляционной природой и содержат потенциально полезные факты, которые могут быть использованы для формирования графов знаний. Однако таблицы, будучи формально структурированными, обычно лишены явной семантики. В частности, заголовок столбца «*2023*» может обозначать как год, так и номер проекта, а ячейка со значением «*Apple*» требует дискурсивного анализа для дифференциации компании или фрукта. Проблема усугубляется различиями, которые присущи различным областям. Так, финансовые отчеты часто используют иерархические заголовки и матричную компоновку таблицы, научные статьи – многоуровневые сноски в заголовках, а веб-таблицы часто содержат объединённые ячейки и другие HTML-теги. Существующие проприетарные решения типа Talend [9], Trifacta [10] и Microsoft Semantic Link [11] в основном полагаются на более простые методы, такие как синтаксический анализ и сопоставление регулярных выражений для обнаружения ограниченного набора семантических типов, что приводит к ошибкам интерпретации. Поэтому разработка новых методов, моделей и программных средств, позволяющих производить семантическую интерпретацию таблиц и извлекать из аннотированных табличных данных конкретные сущности, их характеристики и связи, является перспективной областью научных исследований.

Данная работа является продолжением проекта по разработке методологического и программного обеспечения для автоматической семантической интерпретации (аннотирования) таблиц и извлечения новых фактов из аннотированных табличных данных с последующим пополнением предметно-ориентированных графов знаний в рамках платформы Talisman [12], в частности, рассматривается задача тестирования производительности подхода при извлечении фактов из русскоязычных таблиц с использованием нового набора данных (*benchmark*) – RF-200. Этапы подхода, программный инструментарий и другие детали подробно представлены в работе [13], где также рассмотрен демонстрационный пример формирования предметных графов знаний на основе табличных данных.

Основной вклад данной работы заключается в следующем:

- Впервые опубликован новый набор данных RF-200 (*ru-facts-200*), содержащий таблицы на русском языке для решения задачи извлечения новых фактов из русскоязычных таблиц (*fact extraction*). Таблицы были отобраны из корпуса таблиц Russian Web Tables (RWT) [14] и размечены с использованием средств платформы Talisman. Полученный набор данных обеспечивает основу для разработки

мультиязычных решений, способных обрабатывать информацию с различными лингвистическими особенностями. Более того, кодовая база созданного набора опубликована для свободного использования.

- Прирост экспериментальной оценки (F-меры) производительности предлагаемого подхода к аннотированию таблиц и извлечению новых фактов из аннотированных табличных данных на основе созданного набора данных RF-200 составил 0,187 относительно базового решения (*baseline*) извлечение фактов из текстов.
- Интеграция решения в промышленную платформу Talisman, что подтверждает прикладную значимость исследования.

Статья организована следующим образом: в разделе 2 представлено современное состояние исследований и существующие наборы табличных данных. В разделе 3 кратко описывается предложенный ранее подход к семантическому аннотированию таблиц. В разделе 4 описывается процесс создания нового набора русскоязычных табличных данных RF-200. Раздел 5 представляет результаты тестирования производительности авторского подхода с использованием созданного набора данных. В заключении дается обсуждение полученных результатов и планы будущей работы.

2. Современное состояние исследований

В силу большого распространения табличных данных в последнее время все больше исследователей обращают внимание на проблематику создания (*knowledge graph construction* [15-16]), пополнения (*knowledge graph population* [17-18]) и расширения графов знаний (*knowledge graph refinement* [19]) за счет этой информации.

Недостатки существующих решений обуславливают необходимость разработки новых методов, сочетающих автоматизированную обработку слабоструктурированных таблиц с интуитивными интерфейсами и средствами семантической верификации. Несмотря на определенные успехи, область остаётся фрагментарной: отсутствует универсальная методология, способная обеспечить комплексную интерпретацию разнородных таблиц. Согласно экспериментальным данным последнего соревнования SemTab-2024 (*Semantic Web Challenge on Tabular Data to Knowledge Graph Matching*) [20], современные системы демонстрируют неудовлетворительную точность при работе с реальными данными. Более того, существующие решения, как правило, не предоставляют возможности дальнейшего использования семантически аннотированных таблиц, например, пропуская этап формирования графов знаний. Указанные ограничения подчеркивают необходимость создания интегрированных платформ, способных не только генерировать предметные графы знаний из таблиц, но и динамически расширять существующие семантические модели новыми фактами. Приоритетными направлениями остаются разработка кросс-доменных алгоритмов, внедрение пользовательских графических интерфейсов для экспертов-непрограммистов и обеспечение открытости инструментария.

Для тестирования производительности подходов к автоматическому пониманию табличной информации (*table understanding*) [21-22] используются наборы данных (*benchmarks*), называемые также «*золотыми стандартами*» (*gold standards*), которые служат эталоном для измерения производительности (качества) различных методов и систем. Они позволяют выявлять сильные и слабые стороны существующих методов, тем самым помогая в продвижении производительности на современном уровне. Большинство доступных наборов данных для табличных задач охватывают широкий диапазон предметных областей, в основном за счет того, что создаются с использованием крупномасштабных открытых веб-ресурсов, таких как Wikipedia или GitHub, и только некоторые из них нацелены на определенную конкретную предметную область (например, медицину, финансы, промышленность).

Наборы данных для задачи семантической интерпретацией таблиц представлены в табл. 1. Указанные наборы содержат таблицы, размеченные семантическими типами (классами, характеристиками и связями между классами), взятых из различных графов знаний общего назначения, для оценки качества семантического аннотирования отдельных элементов таблиц. По полученным аннотациям из ячеек таблиц могут быть извлечены факты, однако эти наборы напрямую не направленны на задачу извлечения фактов и не предоставляют каких-либо метрик оценки для этого. Следует также отметить, что таблицы в этих наборах представлены на английском языке, исключение составляет только RWT-RuTaBERT, содержащий коллекцию русскоязычных размеченных таблиц.

Табл. 1. Статистика по наиболее распространенным наборам данных для задачи семантической интерпретации таблиц.
Table 1. Statistics on the most common datasets for the task of semantic table interpretation.

Набор данных	**Кол-во таблиц**	**Кол-во столбцов**	**Кол-во строк**	**Кол-во семантических типов**	**Граф знаний**
Limaye [23]	6,5 тыс.	–	–	837	Wikidata, Yago
T2Dv2 [24]	234	1,2 тыс.	2,8 тыс.	193	DBpedia
Tough Table (2T) [25]	180	194 тыс.	802 тыс.	540	DBpedia, Wikidata
BiodivTab [26]	50	1,2 тыс.	12,9 тыс.	84	Wikidata
GitTables [27]	962 тыс.	11,5 млн.	13,6 млн.	2,4 тыс.	Schema.org, DBpedia
SOTAB [28]	108 тыс.	–	–	267	Schema.org
VizNet-Sato [29]	–	120,6 тыс.	–	78	DBpedia
WikiTabels-TURL [30]	–	628,2 тыс.	–	225	Freebase
RWT-RuTaBERT [31]	–	1,4 млн.	–	170	DBpedia

Тем не менее, существует небольшой ряд примеров наборов данных, ориентированных на задачу извлечения фактов (сущностей) из таблиц, в частности:

- **SWDE (Structured Web Data Extraction)** [32] – структурированный набор данных, извлеченных из 128 000 HTML-страниц с 80 веб-сайтов. Собранные записи распределены по восьми категориям: «*автомобили*», «*книги*», «*камеры*», «*работа*», «*фильмы*», «*игроки Национальная Баскетбольная Ассоциация (НБА)*», «*рестораны*» и «*университеты*». Для каждой категории задано от 3 до 5 атрибутов (например, для категории «книга» это будет «название», «автор», «ISBN-код», «издатель» и «*дата публикации*»), которые можно сопоставить столбцам данных. При этом количество строк соответствует количеству страниц (сущностей, например, конкретных книг). Набор данных содержит разметку (*ground truth*), созданную с помощью регулярных выражений для определенных атрибутов.
- **DISCOMAT** [33] – содержит 5 883 таблиц в формате CSV, извлеченных из 2 536 научных статей по материаловедению из баз Interglad, SciGlass и Elsevier. При этом

1 475 таблиц были размечены вручную, остальные автоматически. Набор включает четыре основных семантических типа сущностей: «*материал*», «*компонент*», «*процент*» и «*единица измерения*».

- **arXiv Machine Learning Tables** [34] – содержит 122 таблицы в формате LaTeX, извлеченных из 25 научных статей на arXiv по тематике машинного обучения. Набор включает 3 792 аннотированных записей, принадлежащих одиннадцати типам (например, «*метрика*», «*задача*», «*обучающие данные*»).
- **PubMed Chemistry Tables** [34] – содержит 26 таблиц в формате XML, извлеченных из 16 научных статей на PubMed по тематике физических свойств химических соединений. Набор включает записи, принадлежащие трем основным типам: «*единицы измерения*», «*исследуемое соединение*», и «*биологический объект*».

Основная статистика по данным наборам таблиц представлена в табл. 2.

Рассмотренные наборы в основном охватывают относительно простые варианты таблиц, обладающих реляционной природой. Как правило, они содержат небольшое количество семантических типов, относящиеся к какой-то конкретной области или небольшому набору областей. Кроме того, данные в этих таблицах представлены исключительно на английском языке, что ограничивает применение методов для других языков, включая русский. Таким образом, создание новых мультиязычных наборов данных для задачи извлечения фактов из таблиц, относящихся к разнообразным предметным областям и обладающих сложной структурной компоновкой, является актуальным.

Табл. 2. Статистика наборов данных для задачи извлечения фактов из таблиц.
Table 2. Statistics of datasets for the task of fact extraction from tables.

Набор данных	**Кол-во таблиц**	**Кол-во записей**	**Формат**	**Кол-во категорий (типов)**
SWDE	*128 000*	–	HTML	8 категорий: «автомобили», «книги», «камеры», «работа», «фильмы», «игроки Национальная Баскетбольная Ассоциация (НБА)», «рестораны» и «университеты»
DISCOMAT	*5 883*	–	CSV	4 типа: «материал», «компонент», «процент» и «единица измерения»
arXiv Machine Learning Tables	*122*	3 792	LaTeX	11 типов: «метрика», «задача», «обучающие данные» и др.
PubMed Chemistry Tables	*26*	1 498	XML	3 типа: «единицы измерения», «исследуемое соединение», и «биологический объект»

3. Существующий задел

Разработанный авторами подход реализует семантическое аннотирование колонок и отношений между ними, которое заключается в сопоставлении колонкам релевантных *типов характеристик*, определении наиболее подходящего *типа концепта* на их основе, а также выявление *типов связей* между определенными типами концептов. После установления подобной аннотации из строк таблиц последовательно извлекаются новые факты и добавляются в целевой граф знаний. Обобщенная схема подхода приводится на рис. 1.

Рис. 1. Обобщенная схема подхода.
Fig. 1. The scheme of the approach.

Подход состоит из четырех основных этапов:

1) **Предобработка таблиц:** Модель XLM-RoBERTa, дообученная на корпусах CoNLL-2003, OntoNotes и DocRED, выполняет распознавание именованных сущностей (персоны, организации, локации и др.) в ячейках таблицы. На основе NER-меток извлекаются базовые факты (*текстовые упоминания* и *значения характеристик*). Важно отметить, что данная модель по умолчанию доступна в форме специального семантического анализатора, входящего в платформу Talisman. Описание гиперпараметров и другие детали модели представлены в работе [35].
2) **Поиск типов кандидатов:** Для каждой колонки таблицы определяется набор возможных *типов характеристик* из *KG*, исключая колонки без извлеченных базовых фактов.
3) **Аннотирование колонок:** Релевантный тип для колонки выбирается с помощью агрегированной оценки, полученной на основе применения комбинации трех эвристических метода (*голосование большинством, сходство по заголовку, группировка характеристик*). Данная оценка определяет итоговую вероятность того, что определенный тип характеристики из набора кандидатов является наиболее подходящим (релевантным) для аннотирования столбца таблицы. Агрегирование осуществляется на основе линейной свертки оценок, полученных каждой эвристикой: $f_{agg}(c_i) = f_1(c_i) \times w_1 + f_2(c_i) \times w_2 + f_3(c_i) \times w_3$, где c_i – целевой столбец для аннотирования; f_1, f_2, f_3 – эвристики аннотирования столбца; w_1, w_2, w_3 – весовые коэффициенты, которые уравновешивают важность каждой оценки.
4) **Извлечение и добавление фактов в целевой граф знаний:** На основе аннотаций извлекаются *концепты*, их *характеристики* и *связи*, пополняя граф знаний Talisman. При этом факты связей формируются только в пределах одной строки.

Разработанный подход реализован в форме специального автоматического *обработчика таблиц* (*tables-annotator*), который использовался в рамках исследовательского проекта Института системного программирования имени Иванникова Российской академии наук (ИСП РАН). В рамках этого проекта решалась задача автоматизированного наполнения предметно-ориентированных графов знаний платформы Talisman [12] новыми фактами, извлеченными, в том числе, из табличных данных. Подробная информация по подходу представлена в работе [13].

4. Набор данных RF-200

Структурная компоновка таблиц и большой охват разнообразных предметных областей важны для создания качественного набора табличных данных. В данной работе использовался крупномасштабный корпус табличных данных – Russian Web Tables (RWT) [14], который был сформирован на основе среза русскоязычной Википедии за 13 сентября 2021 года. RWT представлен набором файлов в формате CSV, содержащих непосредственно таблицы, а также файлы в формате JSON, содержащие метаданные о таблицах. В табл. 3 описывается основная статистика корпуса RWT.

Табл. 3. Статистика корпуса таблиц RWT.
Table 3. The RWT corpus statistics.

Статистика	**Значение**
Количество таблиц	1 266 731
Количество колонок	7 419 771
Количество ячеек	99 638 194
Среднее число ячеек на таблицу	81,78
Размер набора	17 ГБ
Процент практически пустых колонок	6%
Среднее число ячеек в колонке	13,42
Процент колонок содержащих только числовые данные	17%

Из корпуса RWT было отобрано 225 исходных вертикальных таблиц, в которых данные содержаться в форме столбцов (вертикальных колонок), на основе метаданных корпуса. Собранные таблицы принадлежат 26 разным предметным областям и содержат как простые заголовки (заголовок первой строкой), так и сложные иерархические заголовки, которые могут располагаться в произвольном порядке внутри таблицы. Данные в таблицах не были очищены и могут содержать незначительные опечатки и мусорные символы, которые потенциально могут вносить сложность в процесс обработки и анализа этих таблиц.

Разметка данных таблиц осуществлялась в автоматизированном режиме средствами платформы Talisman [12]. В частности, была разработана модель предметной области (*OntoScheme*), отражающая основные понятия, их характеристики и отношения между ними для всех 26 областей. Статистика по созданной модели представлена в табл. 4.

Пример фрагмента модели предметной области, описывающий область музыки (данные по чартам, синглам, информация об исполнителях и музыкальных группах и т.п.), представлен на рис. 2.

Затем применялся специальный обработчик извлечения фактов, к которому вручную определялась конфигурация в формате JSONPath отдельно для каждой таблицы. Данная конфигурация представляет собой набор инструкций, использующий механизм регулярных выражений и созданную модель предметной области, по которым происходило извлечение фактов из таблиц. Извлеченные таким образом факты составили эталонные данные разметки (*ground truth*). В результате было размечено 200 таблиц. Статистика по собранным и размеченным таблицам представлена в табл. 5.

В результате был создан новый набор данных – RF-200 (*ru-fats-200*), содержащий размеченные русскоязычные таблицы. Основная статистика по набору RF-200 представлена

в табл. 6. При этом среднее количество колонок на одну таблицу составило 4,89. Среднее количество ячеек на 1 таблицу составило 97,45, а доля пустых ячеек 8,82%.

Табл. 4. Статистика по созданной модели предметной области.
Table 4. Statistics on the created domain model.

Элемент модели	**Количество**
Тип концепта	29
Тип значения характеристики	9
Тип характеристики концепта	281
Тип связи	50
Тип характеристики связи	22
Всего элементов	**391**

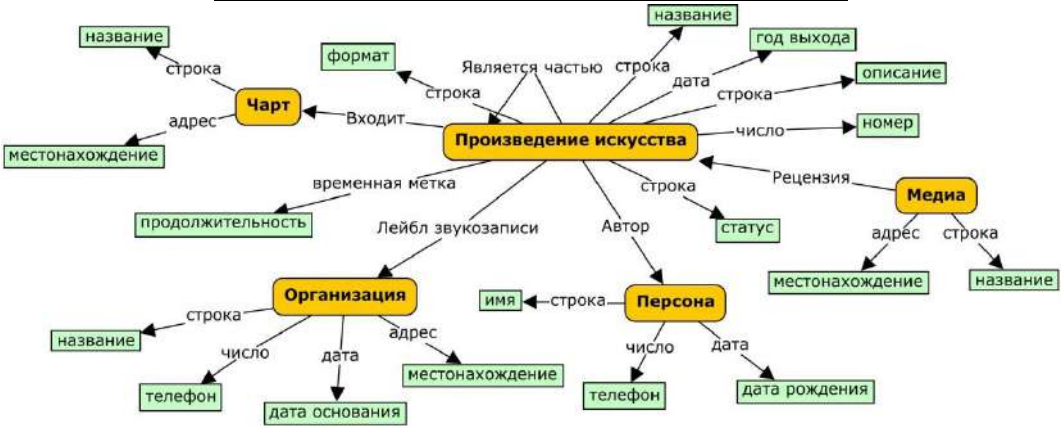


Рис. 2. Пример фрагмента модели предметной области, описывающий область «музыка».
Fig. 2. An example of a domain model fragment ("music").

4. Тестирование производительности

4.1 Настройки и метрики

Тестирование производительности авторского подхода и его программной реализации в форме обработчика платформы Talisman осуществлялась на основе подготовленного набора данных RF-200. В качестве базового решения (*baseline*) для сравнения был выбран классический подход извлечения фактов из текстов, основанный на распознавании именованных сущностей (*Named Entity Recognition*) и извлечении отношений между ними (*Relation Extraction*). Этот подход также реализован в форме специального обработчика – *семантического анализатора* (*semantic analyzer*) в платформе Talisman. Следует отметить, что провести корректное сравнение с другими внешними "*state-of-the-art*" решениями достаточно сложно, так как они направлены на обработку только определенного набора данных, обладающих собственной спецификой (форматом, поддерживаемых категорий типов и др.).

Основная проверяемая гипотеза заключается в ответе на следующие вопросы: «*Пригодны ли классические методы извлечения информации из текстов для таблиц?*» и «*Требуют ли таблицы создания специализированных решений?*».

Табл. 5. Статистика по собранным и размеченным таблицам.
Table 5. Statistics on the collected and labeled tables.

№	Предметная область	Краткое описание	Кол-во отобранных таблиц	Кол-во размеченных таблиц
1	*Локации*	Статистика по странам, отдельным субъектам, городам (население,	34	33
2	*Спорт*	Команды, игроки, виды спорта	29	26
3	*Киноиндустрия и театры*	Фильмы, сериалы, аниме, театральные постановки, актеры	18	16
4	*Политика*	Партии, депутаты, политики, лидеры	16	15
5	*Кинонаграды*	Кинопремии, победители, номинации и	11	8
6	*История*	События, личности и военная статистика	9	9
7	*Музыка*	Песни, синглы, певцы, группы	9	6
8	*Автоспорт*	Ралли, команды, турниры	8	8
9	*Архитектурные*	Строения как старые, так и новые	8	8
10	*Торговля и финансы*	ВВП, кредиты, импорт, экспорт	7	3
11	*Энергетика*	Показатели энергетики, мощности	7	2
12	*Печатные издания*	Книги, манга, журналы	7	7
13	*Измерения*	Эталонные и рекордные измерения различных показателей	6	6
14	*Праздники и мероприятия*	Названия праздников, периоды празднования	6	6
15	*Природные объекты*	Статистика по рекам и озерам	6	6
16	*Авиация*	Самолеты, вертолеты	5	5
17	*Медиа*	Радио, телевидение	5	5
18	*Организации и объединения*	Данные по различным организациям и объединениям	5	5
19	*Продукты питания*	Статистика по составу продуктов питания	4	4
20	*Астрономия*	Астрономические аппараты, звёздные	4	4
21	*Национальности и этносы*	Статистические данные по различным национальностям, этносам и	4	4
22	*Религия*	Статистические данные по различным конфессиям и религиозным течениям	4	4
23	*Телешоу*	КВН, стендап, команды	4	1
24	*Награды и премии*	Статистические данные по различным номинациям, премиям и награжденным	3	3
25	*Реслинг*	Соревнования по реслингу, статистика по победам рестлеров	3	3
26	*Компьютерные игры*	Игры, платформы, игровые издания, киберспорт	3	3
ИТОГО			**225**	**200**

Табл. 6. Статистика по размеченному набору табличных данных RF-200.
Table 6. Statistics on the labeled tabular dataset (RF-200).

Предметная область	**Кол-во таблиц**	**Кол-во колонок**	**Кол-во ячеек**	**Кол-во пустых ячеек**
Локации	33	153	1992	99
Спорт	26	188	4786	614
Киноиндустрия и театры	16	74	1464	44
Политика	15	68	915	81
История	9	33	877	156
Кинонаграды	8	42	588	3
Автоспорт	8	45	431	10
Архитектурные сооружения	8	47	753	124
Печатные издания	7	29	554	48
Музыка	6	21	618	6
Измерения	6	23	248	11
Праздники и мероприятия	6	20	263	12
Природные объекты	6	26	359	21
Авиация	5	28	499	109
Медиа	5	16	221	17
Организации и объединения	5	37	640	55
Продукты питания	4	14	194	0
Астрономия	4	16	800	0
Национальности и этносы	4	12	243	10
Религия	4	11	102	0
Торговля и финансы	3	12	280	0
Награды и премии	3	13	342	52
Реслинг	3	14	472	41
Компьютерные игры	3	14	906	143
Энергетика	2	16	880	63
Телешоу	1	7	63	0
ИТОГО	**200**	**979**	**19490**	**1719**

Интуитивно, метрика оценки должна вычислять разницу между количеством истинных (размеченных) фактов, находящихся в таблице набора RF-200 и количеством извлеченных фактов обработчиком таблиц и семантическим анализатором. Таким образом, экспериментальная оценка была получена отдельно для двух этапов:

1) извлечение *фактов-концептов*, *фактов-значений* и *фактов-упоминаний* (будем обозначать этот этап как «NERC»);
2) извлечение *фактов-характеристик концептов*, *фактов-связей* и *фактов-характеристик связей* (будем обозначать этот этап как «RELEXT»).

В качестве метрик оценки для обоих этапов извлечения фактов использовались стандартные: точность (*precision*), полнота (*recall*) и F-мера (*F1 score*):

$$Precision = \frac{CF}{EF}, \qquad Recall = \frac{CF}{NF}, \qquad F1 = \frac{2 \times Precision \times Recall}{Precision + Recall},$$

где *CF* – количество правильно (т.е. совпадающих с истинными) извлеченных фактов из таблицы обработчиком; *EF* – количество фактов в целом, извлеченных обработчиком из таблицы; *NF* – общее количество фактов, содержащиеся в таблице набора RF-200.

Таким образом, данные метрики считались для каждой таблицы и потом суммировались для всего набора.

4.2 Результаты

Итоговые результаты тестирования производительности извлечения фактов из таблиц на наборе данных RF-200 приведены в табл. 7.

Табл. 7. Результаты экспериментальной оценки на наборе RF-200.
Table 7. The results of experimental evaluation on the RF-200 dataset.

Этап	Семантический анализатор			Обработчик таблиц		
	Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**
NERC	0,668	0,542	0,554	0,659	0,641	0,623
RELEXT	0,000	0,000	0,000	0,377	0,281	0,306
NERC + RELEXT	0,334	0,271	0,277	0,518	0,461	0,464

Экспериментальная оценка по отдельным предметным областям для NERC-этапа приведена в табл. 8, а для RELEXT-этапа приведена в табл. 9. Далее обсудим ключевые выводы по полученной оценке.

Выводы и замечания по полученной оценки производительности:

- Оценка производилась отдельно по каждой таблице с использованием только определенной части модели предметной области (подмножества типов).
- Точность NERC-этапа оказалась немного выше для семантического анализатора. Это связано с тем, что семантический анализатор может точно выделять необходимые значения в ячейках. В то время как обработчик таблиц всегда выделяет значения ячеек целиком.
- Оценка полноты NERC-этапа оказалась выше для обработчика таблиц за счет того, что семантический анализатор может пропускать некоторые значения ячеек в колонке, особенно если они принадлежат к редким NERC-меткам (например, это могут быть редко встречаемые события, мероприятия, механические системы и т.п.). В то время как обработчик таблиц всегда выделяет все значения ячеек в колонке.

Табл. 8. Экспериментальная оценка по отдельным предметным областям для этапа NERC.
Table 8. The experimental evaluation of selected domains for the NERC stage.

Предметная область	**Семантический анализатор**			**Обработчик таблиц**		
	Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**
Национальности и этносы	0,950	0,854	0,883	0,998	0,937	0,963
Продукты питания	0,815	0,361	0,495	0,845	0,875	0,859
Политика	0,718	0,727	0,708	0,853	0,808	0,812
Медиа	0,703	0,737	0,711	0,768	0,742	0,754
Музыка	0,844	0,632	0,621	0,827	0,792	0,751
Природные объекты	0,819	0,506	0,555	0,893	0,677	0,737
Реслинг	0,767	0,793	0,767	0,737	0,727	0,715
Локации	0,688	0,637	0,602	0,715	0,748	0,707
Праздники и мероприятия	0,625	0,782	0,688	0,649	0,798	0,707
Автоспорт	0,676	0,681	0,657	0,684	0,748	0,700
Спорт	0,580	0,678	0,601	0,547	0,699	0,590
Киноиндустрия и театры	0,669	0,481	0,501	0,561	0,576	0,551
Организации и объединения	0,570	0,617	0,562	0,655	0,779	0,699
Компьютерные игры	0,818	0,596	0,660	0,720	0,727	0,693
История	0,734	0,691	0,699	0,677	0,691	0,673
Энергетика	0,749	0,546	0,607	0,831	0,546	0,647
Печатные издания	0,647	0,662	0,634	0,592	0,727	0,642
Авиация	0,485	0,581	0,499	0,516	0,696	0,580
Торговля и финансы	0,727	0,309	0,414	0,667	0,467	0,524
Религия	0,502	0,406	0,426	0,567	0,505	0,520
Архитектурные сооружения	0,621	0,450	0,477	0,565	0,577	0,513
Награды и премии	0,873	0,373	0,491	0,642	0,389	0,481
Кинонаграды	0,569	0,389	0,415	0,406	0,514	0,428
Телешоу	0,455	0,234	0,309	0,469	0,359	0,407
Астрономия	0,596	0,195	0,258	0,608	0,370	0,392
Измерения	0,187	0,167	0,167	0,134	0,186	0,147
По всем областям	**0,668**	**0,542**	**0,554**	**0,659**	**0,641**	**0,623**

Табл. 9. Экспериментальная оценка по отдельным предметным областям для этапа RELEXT.
Table 9. The experimental evaluation of selected domains for the RELEXT stage.

Предметная область	**Обработчик таблиц**		
	Precision	**Recall**	**F1**
Национальности и этносы	0,750	0,628	0,669
Продукты питания	0,667	0,667	0,667
Политика	0,833	0,526	0,609
Медиа	0,784	0,438	0,534
Музыка	0,627	0,378	0,466
Природные объекты	0,427	0,430	0,422
Реслинг	0,461	0,380	0,407
Локации	0,525	0,297	0,360
Праздники и мероприятия	0,392	0,313	0,342
Автоспорт	0,500	0,250	0,333
Спорт	0,410	0,294	0,325
Киноиндустрия и театры	0,513	0,248	0,321
Организации и объединения	0,312	0,328	0,319
Компьютерные игры	0,448	0,266	0,304
История	0,376	0,271	0,302
Энергетика	0,279	0,312	0,285
Печатные издания	0,198	0,350	0,252
Авиация	0,335	0,205	0,230
Торговля и финансы	0,308	0,154	0,205
Религия	0,197	0,206	0,201
Архитектурные сооружения	0,251	0,165	0,193
Награды и премии	0,111	0,111	0,111
Кинонаграды	0,091	0,091	0,091
Телешоу	0,000	0,000	0,000
Астрономия	0,000	0,000	0,000
Измерения	0,000	0,000	0,000
По всем областям	**0,377**	**0,281**	**0,306**

- Экспериментальные оценки семантического анализатора для этапа RELEXT оказались нулевыми из-за того, что данный обработчик может выделять связи и характеристики только внутри одного текста (ячейки). В то время как обработчик таблиц может выделять связи и характеристики между значениями ячеек разных колонок. Данная оценка наглядно показывает, что классический подход извлечения фактов из текстов слабо применим к табличным данным.
- В целом итоговые оценки (этапы NERC + RELEXT) для предлагаемого подхода (обработчика таблиц) оказалась ожидаемо выше, чем для семантического анализатора за счет своей направленности на обработку таблиц.

Основные причины (проблемы), повлиявшие на не высокую оценку производительности:

- Наличие опечаток и «мусорных» тегов HTML в некоторых значениях ячеек.
- Предлагаемый подход включает этап предварительной обработки таблиц, который основан на результатах распознавания именованных сущностей и извлечении связи (рис. 1). Таким образом, работа обработчика таблиц полностью основана на результатах работы семантического анализатора. Поэтому если семантический анализатор в заданной колонке не нашел NERC-метки, то обработчик таблиц пропустит эту колонку.
- Текущая реализация обработчика таблиц не позволяет извлекать характеристики связей.
- В ячейках с идентифицирующими характеристиками (например, названиями) могут попадаться пустые ячейки или может стоять прочерк или символ «н/д».
- В разных ячейках одной колонки могут быть представлены разные концепты с характеристиками (например, колонка может содержать одновременно как концепты типа «*Персона*», так и «*Организация*»).
- В одной ячейке могут быть представлены разные концепты или характеристики (например, ячейка может содержать регион с его географическими координатами).
- В одной ячейке могут быть представлены множественные значения концептов или характеристик одного типа (например, может быть перечисление имен или организаций).
- Концепт с его идентифицирующей характеристикой (названием) может быть расположен вне таблицы (например, в заголовке), а в самой таблице есть только характеристики этого концепта (например, для песни, название которой вынесено в заголовок, в таблице представлены только продолжительность песни и дата ее записи).
- Названия заголовков являются названиями характеристик концепта или связи.
- Характеристика концепта или связи является составной и распределена в нескольких колонках (например, счет в футбольном матче может быть разбит на несколько ячеек).
- Характеристика концепта или связи является составной и содержится в одной ячейке (например, год, состоящий из диапазона, или карьера игрока, состоящая из множества дат).
- Наличие вычисляемых значений ячеек в колонках (например, расчет времени участников в гонке относительно времени победителя).
- Разные единицы измерения, которые могут приводить к нескольким видам характеристик. Например, в двух таблицах с измерением численности населения указаны «*млн. чел.*» и «*тыс. чел.*».

Для устранения определенных выше проблем требуется улучшить существующие методы семантического аннотирования, в частности, требуется добавить:

- Корректную обработку извлечения идентифицирующих характеристик с учетом пустых ячеек, прочерков или специальных символов.
- Извлечение характеристик связей из таблиц.
- Извлечение фактов составных характеристик из таблиц, которые могут собираться как внутри одной ячейки, так и быть собраны из ячеек разных столбцов.
- Обработку множественных однотипных значений (концептов и характеристик) в ячейках.
- Извлечение фактов из таблиц с использованием внешнего контекста таблицы, в частности, связь с фактами, которые расположены в остальном документе (например, в заголовке названия таблицы).

В целом, полученные результаты показывают перспективность использования разработанного подхода и обработчика таблиц для поддержки процесса извлечения конкретных сущностей (фактов) из семантически аннотированных табличных данных и пополнения ими предметно-ориентированных графов знаний.

5. Заключение

Эффективное тестирование методологического и программного обеспечения для автоматической семантической интерпретации (аннотирования) таблиц и извлечения новых фактов из аннотированных табличных данных требует создания и использования русскоязычных наборов данных.

Основной вклад данного исследования заключается в создании первого русскоязычного набора табличных данных RF-200, охватывающего 26 предметных областей, а также в результатах оценки производительности авторского подхода. Набор опубликован и доступен для свободного использования на GitHub [36]. Программная реализация подхода в форме обработчика платформы Talisman продемонстрировала его превосходство над традиционными методами извлечения фактов из текстов, достигнув F-меры 0.464 на этапах NERC и RELEXT. Полученные результаты свидетельствуют о перспективности использования специализированных решений для работы со структурированными данными, особенно в условиях лингвистического разнообразия.

Результаты исследования имеют как теоретическую, так и практическую значимость. С теоретической точки зрения, предложенный метод аннотирования устраняет субъективность за счёт статистической верификации, что расширяет возможности семантической интерпретации таблиц за пределы числовых данных. С практической точки зрения, созданный набор данных RF-200 позволяет проводить эффективное тестирование производительности современных решений в области обработки таблиц и извлечения фактов. Однако работа выявила ряд ограничений. Во-первых, зависимость от качества распознавания именованных сущностей (NER) может приводить к пропуску колонок с редкими метками. Во-вторых, текущая реализация подхода не поддерживает извлечение характеристик связей и обработку составных значений в ячейках.

Перспективные направления будущих исследований включают интеграцию методов глубокого обучения, основанных на тонкой настройке предварительно-обученных языковых моделей (например, RuTABERT [14]), для повышения точности и автоматизации аннотирования таблиц с более сложной структурой. Созданный набор данных RF-200 будет расширен за счёт включения горизонтальных и матричных таблиц с иерархическими заголовками с объединёнными ячейками, а также поддержку мультиязычности. Кроме того, для подтверждения выводов планируется провести дополнительные статистические тесты,

расчёты доверительных интервалов и измерения межаннотационного согласия на RF-200, с целью определения типов таблиц, для которых предлагаемый подход обеспечивает получение максимальных оценок. Перспективной также является задача обеспечения отображения элементов существующей онтологической схемы графа знаний платформы Talisman в онтологические понятия графов знаний общего назначения такие как Wikidata или DBpedia для увеличения семантической согласованности и упрощения повторного использования созданного набора данных RF-200 за счет предоставления стандартизированной поддержки видов фактов в формате Семантического Веба (RDF/OWL).

Список литературы

[1]. Hogan A., Blomqvist E., Cochez M., d'Amato C., De Melo G., Gutierrez C., Gayo J. E. L., Kirrane S., Neumaier S., Polleres A., Navigli R., Ngomo A.-C. N., Rashid S. M., Rula A., Schmelzeisen L., Sequeda J., Staab S., Zimmermann A. Knowledge Graphs. Springer Nature Switzerland, 2021, 237 p. DOI: 10.1007/978-3-031-01918-0.

[2]. Ji S., Pan S., Cambria E., Marttinen P., Yu P.S. A Survey on Knowledge Graphs: Representation, Acquisition and Applications. IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 2, 2021, pp. 494-514. DOI: 10.1109/TNNLS.2021.3070843.

[3]. 5-star Open Data, Available at: https://5stardata.info/en/, accessed 22.04.2025.

[4]. DBpedia, Available at: https://www.dbpedia.org/, accessed 22.04.2025.

[5]. Wikidata, Available at: https://www.wikidata.org/, accessed 22.04.2025.

[6]. Villazon-Terrazas B., Garcia-Santa N., Ren Y., Srinivas K., Rodriguez-Muro M., Alexopoulos P., Pan J. Z. Construction of Enterprise Knowledge Graphs (I). Exploiting Linked Data and Knowledge Graphs in Large Organisations, Springer, Cham, 2017.

[7]. Number of Google Sheets and Excel Users Worldwide, Available at: https://askwonder.com/research/number-google-sheets-users-worldwide-eoskdoxav, accessed 22.04.2025.

[8]. Peeters R., Brinkmann A., Bizer C. The Web Data Commons Schema.org Table Corpora. Proc. the ACM Web Conference (WWW'24), New York, NY, USA, 2024, pp. 1079-1082. DOI: 10.1145/3589335.3651441.

[9]. Talend, Available at: https://www.talend.com/, accessed 22.04.2025.

[10]. Trifacta, Available at: https://asana.com/ru/apps/trifacta, accessed 22.04.2025.

[11]. Microsoft Semantic Link, Available at: https://learn.microsoft.com/en-us/fabric/data-science/semantic-link-overview, accessed 22.04.2025.

[12]. Talisman, Available at: http://talisman.ispras.ru, accessed 22.04.2025.

[13]. Dorodnykh N. O., Yurin A. Yu. Automated Extraction of Facts from Tabular Data based on Semantic Table Annotation. Trudy ISP RAN/Proc. ISP RAS, vol. 36, no. 3, 2024, pp. 93-104. DOI: 10.15514/ISPRAS-2024-36(3)-7.

[14]. Fedorov P. E., Mironov A. V., Chernishev, G. A. Russian Web Tables: A Public Corpus of Web Tables for Russian Language Based on Wikipedia. Lobachevskii Journal of Mathematics, vol. 44, 2023, pp. 111-122. DOI: 10.1134/S1995080223010110.

[15]. Kruit B., Boncz P., Urbani J. Extracting novel facts from tables for knowledge graph completion. Proc. the 18th International Semantic Web Conference (ISWC'2019), Auckland, New Zealand, 2019, pp. 364-381. DOI: 10.1007/978-3-030-30793-6_21.

[16]. Zhang S., Meij E., Balog K., Reinanda R. Novel entity discovery from web tables. Proc. the ACM Web Conference (WWW'20), New York, NY, USA, 2020, pp. 1298-1308. DOI: 10.1145/3366423.3380205.

[17]. Zhang S., Balog K. Web Table Extraction, Retrieval, and Augmentation: A Survey. ACM Transactions on Intelligent Systems and Technology, vol. 11, no. 2, 2020, pp. 1-35. DOI: 10.1145/3372117.

[18]. Balog K. Populating Knowledge Bases. Entity-Oriented Search INRE, vol. 39, 2018, pp. 189-222. DOI: 10.1007/978-3-319-93935-3_6.

[19]. Subagdja B., Shanthoshigaa D., Wang Z., Tan A.-H. Machine Learning for Refining Knowledge Graphs: A Survey. ACM Computing Surveys, vol. 56, no. 6, 2024, pp. 1-38. DOI: 10.1145/3640313.

[20]. SemTab-2024, Available at: https://sem-tab-challenge.github.io/2024/, accessed 22.04.2025.

[21]. Bonfitto S., Casiraghi E., Mesiti M. Table understanding approaches for extracting knowledge from heterogeneous tables. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 11, no. 4, 2021, e1407. DOI: 10.1002/widm.1407.

[22]. Zheng M., Feng X., Si Q., She Q., Lin Z., Jiang W., Wang W. Multimodal Table Understanding. Proc. the 62nd Annual Meeting of the Association for Computational Linguistics (ACL'2024), Bangkok, Thailand, 2024, pp. 9102-9124. DOI: 10.18653/v1/2024.acl-long.493.

[23]. Limaye G., Sarawagi S., Chakrabarti S. Annotating and searching web tables using entities, types and relationships. Proceedings of the VLDB Endowment, vol. 3, no. 1-2, 2010, pp. 1338-1347. DOI: 10.14778/1920841.1921005.

[24]. T2Dv2 Gold Standard for Matching Web Tables to DBpedia, Available at: https://webdatacommons.org/webtables/goldstandardV2.html, accessed 22.04.2025.

[25]. Cutrona V., Bianchi F., Jimenez-Ruiz E., Palmonari M. Tough tables: Carefully evaluating entity linking for tabular data. Proc. the 19th International Semantic Web Conference (ISWC'2020), Athens, Greece, 2020, pp. 328-343. DOI: 10.1007/978-3-030-62466-8_21.

[26]. Abdelmageed N., Schindler S., Konig-Ries B. Biodivtab: A table annotation benchmark based on biodiversity research data. Proc. the 20th International Semantic Web Conference (ISWC'2021) – Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab-2021), 2021, pp. 13-18.

[27]. Hulsebos M., Demiralp C., Groth P. GitTables: A Large-Scale Corpus of Relational Tables. Proceedings of the ACM on Management of Data, vol. 1, no. 1, 2023, pp. 1-17. DOI: 10.1145/3588710.

[28]. SOTAB (Web Data Commons - Schema.org Table Annotation Benchmark), Available at: https://webdatacommons.org/structureddata/sotab/, accessed 22.04.2025.

[29]. Zhang D., Suhara Y., Li J., Hulsebos M., Demiralp C., Tan W.-C. Sato: Contextual semantic type detection in tables. Proc. the VLDB Endowment, vol. 13, no. 11, 2020, pp. 1835-1848. DOI: 10.14778/3407790.3407793.

[30]. Deng X., Sun H., Lees A., Wu Y., Yu C. TURL: Table Understanding through Representation Learning. Proc. the VLDB Endowment, vol. 14, no. 3, 2020, pp. 307-319. DOI: 10.14778/3430915.3430921.

[31]. Tobola K. V., Dorodnykh N. O. Semantic Annotation of Russian-Language Tables Based on a Pre-Trained Language Model. Proc. the 2024 Ivannikov Memorial Workshop (IVMEM), 2024, pp. 62-68. DOI: 10.1109/IVMEM63006.2024.10659709.

[32]. Hao Q., Cai R., Pang Y., Zhang L. From one tree to a forest: a unified solution for structured web data extraction. Proc. the 34th international ACM SIGIR conference on Research and development in Information Retrieval, Beijing, China, 2011, pp. 775-784. DOI: 10.1145/2009916.2010020.

[33]. Gupta T., Zaki M., Khatsuriya D., Hira K., Krishnan N. M. A., Mausam. DISCOMAT: Distantly Supervised Composition Extraction from Tables in Materials Science Articles. Proc. the 61st Annual Meeting of the Association for Computational Linguistics (ACL'2023), Toronto, Canada, 2023, pp. 13465-13483. DOI: 10.18653/v1/2023.acl-long.753.

[34]. Bai F., Kang J., Stanovsky G., Freitag D., Dredze M., Ritter A. Schema-Driven Information Extraction from Heterogeneous Tables. Proc. the 61st Annual Meeting of the Association for Computational Linguistics (ACL'2024), Miami, Florida, USA, 2024, pp. 10252-10273. DOI: 10.18653/v1/2024.findings-emnlp.600.

[35]. Conneau A., Khandelwal K., Goyal N., Chaudhary V., Wenzek G., Guzmán F., Grave E., Ott M., Zettlemoyer L., Stoyanov V. Unsupervised Cross-lingual Representation Learning at Scale. Proc. the 58th Annual Meeting of the Association for Computational Linguistics (ACL'2020), 2020, pp. 8440-8451. DOI: 10.18653/v1/2020.acl-main.747.

[36]. RF-200 (ru-facts-200), Available at: https://github.com/YRL-AIDA/ru-facts-200, accessed 22.04.2025.

Информация об авторах / Information about authors

Никита Олегович ДОРОДНЫХ – кандидат технических наук, старший научный сотрудник Института динамики систем и теории управления им. В.М. Матросова Сибирского отделения РАН (ИДСТУ СО РАН) с 2021 года. Сфера научных интересов: автоматизация создания интеллектуальных систем и баз знаний, получение знаний на основе преобразования концептуальных моделей и электронных таблиц.

Nikita Olegovych DORODNYKH – Cand. Sci. (Tech.), senior associate researcher at Matrosov Institute of System Dynamics and Control Theory named SB RAS (ISDCT SB RAS) since 2021. Research interests: computer-aided development of intelligent systems and knowledge bases, knowledge acquisition based on the transformation of conceptual models and tables.

Александр Юрьевич ЮРИН – доктор технических наук, заведующий лабораторией Информационно-телекоммуникационных технологий исследования природной и техногенной безопасности ИДСТУ СО РАН, профессор Института информационных технологий и анализа данных Иркутского научно-исследовательского технического университета (ИрНИТУ). Его научные интересы включают разработку систем поддержки принятия решений, экспертных систем и баз знаний, использование прецедентного подхода и семантических технологий при проектировании интеллектуальных диагностических систем.

Alexander Yurievich YURIN – Dr. Sci. (Tech.), Head of a laboratory “Information and telecommunication technologies for investigation of natural and technogenic safety” at ISDCT SB RAS and professor of the Institute of information technologies and data analysis of Irkutsk National Research Technical University (INRTU). His research interests include development of decision support systems, expert systems and knowledge bases, application of the case-based reasoning and semantic technologies in the design of diagnostic intelligent systems, maintenance of reliability and safety of complex technical systems.

DOI: 10.15514/ISPRAS-2025-37(5)-17

Comparative Analysis of Requirements Prioritization Methods for Personalized Nutrition Web Applications

A.S. Mozhegova, ORCID: 0009-0009-2533-2750 <asmozhegova@edu.hse.ru>
V.V. Lanin, ORCID 0000-0002-0650-2314 <vlanin@hse.ru>

HSE University,
38, Studencheskaia St., Perm, 614070, Russia.

Abstract. This study investigates the application of five requirements prioritization methods – MoSCoW, Kano Model, Weighted Scoring, RICE, and Cost of Delay (CoD) – in the development of a web application for personalized nutrition. The research addresses the challenge of managing limited resources (time, financial, and human) while maximizing user value and ensuring safety in a high-stakes domain. Through a comparative analysis, the strengths and weaknesses of each method are evaluated, revealing that a hybrid approach, tailored to different development phases, is most effective. Core functionalities such as allergen management and diet personalization consistently ranked as high priority across all methods. The study proposes a dynamic framework that integrates MoSCoW and Weighted Scoring for MVP definition, and RICE and Kano for scaling, emphasizing the importance of balancing safety, user satisfaction, and implementation complexity. The findings offer practical recommendations for developers and product managers in health-tech and other regulated domains.

Keywords: requirements prioritization; MoSCoW; Kano model; weighted estimation; RICE; Cost of Delay; personalized nutrition; web application.

For citation: Mozhegova A.S., Lanin V.V. Comparative analysis of prioritization techniques in the development of medical web applications. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 225-240. DOI: 10.15514/ISPRAS-2025-37(5)-17.

Acknowledgements. The study was supported by the National Research University as part of the development of a software product.

Сравнительный анализ методов приоритезации требований для веб-приложений персонализированного питания

А.С. Можегова, ORCID: 0009-0009-2533-2750 <asmozhegova@edu.hse.ru>
В.В. Ланин, ORCID: 0000-0002-0650-2314 <vlanin@hse.ru>

Национальный исследовательский университет «Высшая школа экономики», Россия, 614070, г. Пермь, ул. Студенческая, д. 38.

Аннотация. В данном исследовании рассматривается применение пяти методов приоритезации требований – MoSCoW, модели Кано, взвешенной оценки, RICE и стоимости задержки (CoD) – при разработке веб-приложения для персонализированного питания. В исследовании рассматривается проблема управления ограниченными ресурсами (временными, финансовыми и человеческими) при одновременном максимизации ценности для пользователей и обеспечении безопасности в области с высокими ставками. В ходе сравнительного анализа были оценены сильные и слабые стороны каждого метода, что показало, что наиболее эффективным является гибридный подход, адаптированный к различным этапам разработки. Основные функциональные возможности, такие как управление аллергенами и персонализация рациона, неизменно занимают приоритетное место во всех методах. В исследовании предложена динамическая структура, объединяющая MoSCoW и Weighted Scoring для определения MVP, а также RICE и Kano для масштабирования, что подчеркивает важность баланса между безопасностью, удовлетворенностью пользователей и сложностью реализации. Полученные результаты содержат практические рекомендации для разработчиков и менеджеров продуктов в сфере здравоохранения и других регулируемых областях.

Ключевые слова: приоритезация требований; MoSCoW; модель Кано; взвешенная оценка; модель RICE; стоимость задержки; персонализированное питание; веб-приложение.

Для цитирования: Можегова А.С., Ланин В.В. Сравнительный анализ методов приоритезации в разработке медицинских веб-приложений. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 225–240 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)–17.

Благодарности: Исследование выполнено при поддержке Национального исследовательского университета Высшая школа экономики в рамках разработки программного продукта.

1. Introduction

Modern lifestyles have increased the emphasis on healthy eating, and people are seeking personalized diets to meet health, fitness and allergy restriction goals. However, creating balanced meal plans that cater to unique dietary requirements remains a time-consuming and complex process. Personalized nutrition apps are designed to simplify this process by offering customized recommendations based on individual preferences, budget, and health restrictions.

Although the concept of personalized nutrition is not new, the prioritization of requirements for such applications, especially in resource-limited settings, has not been systematically studied. Existing studies often overlook trade-offs between safety (e.g., allergen management), user satisfaction, and implementation complexity. This gap is critical because improper prioritization can lead to development delays, budget overruns, or unmet user needs.

Despite extensive research on requirements prioritization techniques, the existing literature lacks specialized mechanisms that address the unique constraints of medical applications. Traditional approaches, such as MoSCoW or weighted evaluation, either oversimplify subject-specific requirements (e.g., treating allergen filtering as cosmetic user interface improvements) or require impractical data collection (e.g., extensive Kano user surveys). Three critical shortcomings are identified, namely that neither method systematically balances health safety imperatives with user satisfaction metrics. The second problem is that existing methods do not adapt the prioritization logic at different stages of development (MVP vs. scaling). The last drawback is that hybrid approaches remain under-tested in niche areas where regulatory and ethical constraints affect prioritization.

This solution is a hybrid phase platform for medical technology that integrates security and user interaction. The MVP is defined using MoSCoW and weighted scoring: critical features (e.g., allergen warnings) are assigned an increased weight based on risk (anaphylaxis is scored 5× above preference filters). In the scaling phase, RICE is augmented with Kano metrics: if the data confirm that "seasonal recipes" (Kano enjoyment) increase retention, they are assigned a higher priority. Cost of Delay takes into account not only commercial risks (user churn), but also legal risks (late implementation of alerts).

2. Motivation

The complexity of the subject area and limited resources pose significant challenges in the development of personalized nutrition and medical technology solutions. Critical safety requirements such as allergen filtering and medical contraindication verification are imperative, as even a single mistake can jeopardize users' lives, exposing developers to serious legal risks. Studies show that 42% of allergic reactions are caused by hidden allergens (Galland, 2016), making prioritization of safety features absolutely essential over traditional user experience improvements [1].

Dynamic user needs complicate the development process, as personalized nutrition requires constant adaptation to changing trends (e.g., keto, veganism), individual health goals, and budgetary constraints. According to market data, 68% of users prioritize allergen management, while 52% focus on customization for their personalized goals (Market Research Report, 2023) [2].

Regulatory and ethical aspects are an additional challenge. Health technology apps have to comply with stricter standards such as WHO guidelines as opposed to conventional apps. A seemingly "simple" function such as meal planning becomes extremely complex when integrated with real-time allergen databases or dietary recommendations, requiring careful data validation and regulatory compliance. All of this makes developments in this area particularly resource-intensive and high-risk, but critical to ensure the safety and health of users.

Prioritization methods are important in this context because existing methods often fail to meet the unique requirements of medical technology development. Universal frameworks such as MoSCoW treat safety-critical features with the same prioritization logic as cosmetic UI changes, potentially underestimating life-critical requirements. Meanwhile, purely quantitative models such as RICE face limitations due to a lack of early data, especially when it comes to health-specific metrics such as "risk severity" or "likelihood of adverse outcomes". Hybrid approaches, although promising, are rarely tested in high stakes environments where untimely implementation of features such as allergen alerts or drug interaction warnings can lead to legal consequences rather than just user dissatisfaction or churn [3].

This study aims to bridge these gaps by adapting prioritization methods to the specific needs of medical technology. For example, it proposes weighted scoring models with subject-specific multipliers (e.g., 5× weighting for anaphylaxis risk) to ensure that critical safety features do not lose priority due to generic scoring systems. In addition, the study presents an incremental approach to adaptability, combining the simplicity of MoSCoW for MVP development with the scalability of RICE at later stages. This ensures that non-negotiable security requirements are prioritized upfront, while allowing for iterative refinement of UX improvements [4-5].

To validate these methods, the study uses real health data, such as WHO allergy prevalence statistics, to approximate user needs without relying on costly and time-consuming Kano surveys. By basing prioritization on empirical data, the study provides a more robust and scalable framework for health technology product development, ensuring that critical features are implemented with the necessary urgency while maintaining flexibility for continuous improvement.

Although the case study focuses on personalized nutrition, similar challenges arise in medicine (e.g., treatment monitoring) and fintech (regulatory compliance). This makes the proposed method a versatile tool for resource-intensive projects.

3. Related Works

The field of requirements prioritization has undergone significant changes to address software development challenges in various domains. While existing techniques provide a sound foundation for general applications, their adaptation to specialized areas such as medical technology and, in particular, personalized nutrition, reveals critical gaps that this research aims to address. Let us review the most common requirements prioritization techniques.

The MoSCoW method is one of the most popular approaches to requirements prioritization. Widely used in agile environments, MoSCoW categorizes requirements into "Must-have," "Should-have," "Could-have," and "Will not-have." Its simplicity facilitates rapid decision-making, but its reliance on subjective stakeholder input often overlooks risks specific to the subject area. For example, in medical applications, a "Must Have" feature such as allergen filtering may be incorrectly prioritized despite its potential to save lives. Recent adaptations combine MoSCoW with quantitative safety metrics (e.g., severity of health risks) to mitigate this bias, as demonstrated in clinical software projects [6-7].

Kano's model proposes to classify requirements based on their impact on user satisfaction. This framework categorizes features based on their impact on user satisfaction (Basic, Performance, Delighters). While effective for consumer applications, Kano's reliance on extensive user surveys is impractical for niche areas such as personalized nutrition where early data is scarce. Hybrid approaches, such as integrating Kano with WHO health statistics to approximate "basic" needs (e.g., allergen alerts), offer a workaround, but lack validation in the context of health technologies [8-9].

The weighted evaluation method assigns weights to requirements based on criteria such as user value, implementation complexity, and business impact, which allows prioritization of goals. However, it struggles to balance health-specific factors (e.g., regulatory compliance) with conventional metrics. For example, a feature with moderate user value but high legal risk (e.g., allergy alerts) may be undervalued. Recent proposals supplement weights with safety multipliers (e.g., 5× for critical health risks), although empirical validation remains limited.

The RICE method evaluates characteristics using four factors: coverage, impact, confidence, and effort [10]. Developed for product stewardship, RICE assesses coverage, influence, confidence and effort. Its quantitative nature is useful but requires robust data that is often unavailable in the early stages of healthcare projects. Adaptations narrow "coverage" to at-risk users (e.g., allergy sufferers) and include safety as an impact multiplier. However, these adjustments are not tested in personalized nutrition, where dynamic user needs (e.g., dietary trends) further complicate evaluations.

The Cost of Delay (CoD) method quantifies the cost of delaying the realization of a function [11]. This method quantifies the urgency of feature delivery, typically focusing on commercial metrics such as user churn. In medical technology, delaying features may incur legal costs (e.g., non-compliance with FDA guidelines) or health risks (e.g., delayed allergen testing). Simplified versions prioritize features based on two factors: severity of health risk and regulatory deadlines, but lack integration with other methods [12].

4. Problem Statement

Although existing prioritization methods (MoSCoW, Kano, Weighted Scoring, RICE, Cost of Delay) provide generalized frameworks, they do not account for trade-offs specific to medical technology. A key gap is the safety and satisfaction dilemma: current tools apply similar logic to critical functions (e.g., allergen alerts) and convenience functions (e.g., meal planning). For example, MoSCoW may categorize both as "must-haves" despite their very different risk profiles, which is a glaring omission. This confusion can lead to safety-critical functions being prioritized with insufficient respect.

Phase-Ignorant Prioritization is another flaw: most frameworks use static criteria throughout all phases of development, ignoring changing priorities. Early MVP phases require a focus on security (e.g., anaphylaxis prevention), while scaling phases require a focus on user retention (e.g., offering

seasonal recipes). Methods such as RICE and Cost of Delay lack mechanisms to adapt their metrics (e.g., "coverage" or "time sensitivity") to these transitions, resulting in shifting roadmaps.

Finally, the scarcity of data in niche domains hampers methods based on user input (e.g., Kano surveys) or precise estimates (e.g., RICE coverage/impact). Early user data is often lacking in the health technology domain, but available proxies (e.g., WHO allergy statistics) remain underutilized in prioritization models. Without subject-specific adaptations, these frameworks may inadvertently prioritize high stakes features or over-invest in low-impact features.

Two key criteria will be used to evaluate the effectiveness of the chosen approach. The first is resource efficiency, which is defined by the extent to which the method allows rational management of time, financial and human resources [17]. The second is flexibility and adaptability, reflecting the ability of the approach to respond quickly to changes in development requirements and conditions [18].

The study will propose a hybrid framework that integrates weighted safety multipliers (e.g., 5× criticality scores for allergen characteristics) in MoSCoW and a weighted score for MVP, ensuring that high-priority safety features are highlighted early on. After MVP, the system will combine RICE's focus on ROI with Kano satisfaction scores using public health data (e.g., FDA allergen databases, WHO nutritional recommendations) to compensate for reliance on early user surveys.

5. Overview

Developing a web application for personalized nutrition involves balancing user needs such as dietary restrictions, health goals, and budgetary constraints with the challenges of limited resources (time, budget, and personnel). While the architectural components (user interface, business logic, database, and external integrations) may seem standard, the critical issue is domain-specific prioritization, which directly impacts user safety, regulatory compliance, and long-term adherence.

The paper presents domain-specific adaptations such as weighted safety multipliers, e.g., applying 5× criticality scores to allergen-related features to ensure that life-critical requirements are prioritized. It also integrates empirical data such as WHO allergy statistics to reduce reliance on early user surveys and improve decision accuracy. The approach uses phase prioritization: in the MVP phase, it combines MoSCoW and weighted evaluation to focus on security and core functionality, while in the scaling phase it augments RICE with Kano metrics to improve user satisfaction and ROI. Beyond the immediate application, the study offers generalized insights to demonstrate how hybrid prioritization techniques can effectively balance security, user value, and resource efficiency in a capacity-constrained environment. It also provides a reproducible framework for other niche areas with strict regulatory or ethical constraints, such as medical or financial applications.

Key development steps include: requirements gathering (allergen tracking, fitness integration, diet support) [22], prioritization (MoSCoW, Kano, RICE methods) [23], development/testing, and evaluation of prioritization performance [24].

6. Implementation

6.1 Basic Requirements and Identification

A web application for personalized nutrition consists of four main components: a user interface for entering dietary preferences, allergens, and goals [25]; business logic that processes data and generates personalized menus; a database that stores user profiles and product information; and integration of external services with fitness trackers to improve recommendations. This architecture provides efficient data management and personalized meal planning designed around four key business requirements derived from market research.

By addressing both the specific case of personalized nutrition and the broader challenges of health-tech development, this study bridges the gap between theoretical prioritization methods and

practical, high-stakes applications. The proposed framework offers actionable recommendations for developers and product managers, emphasizing dynamic, data-driven decision-making.

The primary requirement is diet personalization based on current trends in dietetics. Studies show that personalized recommendations increase dietary adherence by 37% compared to generic meal plans [26]. Taking individual metabolic characteristics into account is also crucial – for example, a study by Zeevi et al. (2015) proved that the same foods have different effects on blood sugar levels in different people [27]. The market analysis confirmed the demand: 68% of surveyed users wanted to take allergies into account, while 52% were aiming for individualized dietary goals [28].

The second key requirement is safety. According to WHO (2021), 10% of people experience food poisoning each year and 5% of adults have allergies [29].

Allergy Solutions (Galland, 2016) notes that 42% of allergic reactions are caused by hidden allergens [30]. The success of the AllergyEats app has demonstrated that automatic allergen filtering reduces risks by 90% [31].

Another key aspect is to simplify meal planning and shopping. USDA data (2020) shows that families spend 5.6 hours per week on these tasks [32], while the Smarter Faster Better study (Duhigg, 2016) proves that automation can increase productivity by 20-30% [33]. Competitor analysis confirms the need for prescription and shopping integration [34].

Budget control is equally important – 60% of Americans overpay for food (BLS, 2022) [35]. As shown in Eat Well for $4 a Day (Brown, 2015), conscious food choices reduce costs by 15-25% [36]. Platforms such as Budget Bytes demonstrate users' preference for detailed spending analytics.

Based on AS IS analysis, the system should provide personalized dietary adaptation, allergen management, and health goal tracking. It should analyze nutrients, flag risks, and provide personalized recommendations. Features should include smart shopping lists, balanced meal planning (nutritionally and seasonally appropriate) and budget tracking. The platform should support family profiles, real-time pricing, flexible meal replacement, offline access, customizable interfaces, and data export.

6.2 A Comparative Analysis of Prioritization Frameworks

The MoSCoW method serves as a foundational requirements prioritization framework in Agile and product management, offering a structured approach to categorizing features based on their criticality to product success. In the context of health-tech applications—particularly our personalized nutrition web app—this method takes on added significance due to the domain's unique safety, regulatory, and ethical constraints. The acronym MoSCoW delineates four priority tiers: Must-have (M), Should-have (S), Could-have (C), and Won't-have (W), each playing a distinct role in resource-constrained development environments.

For our health-focused application, Must-have requirements were at the core and included features whose absence would make the product unsafe, non-compliant, or fundamentally non-functional. These included critical safety features such as real-time allergen screening (automatic detection of recipes containing user-specified allergens), medical contraindication screening, and basic personalization capabilities (diet type selection). Notably, these features were given absolute priority not only because of their value to users, but also because their absence could lead to serious health consequences or regulatory non-compliance. For example, whereas in a social media app, “push notifications” could be categorized as a Should-have, in our context “allergen alerts” became a Must-have because of their potential to save lives, reflecting the method's adaptation to the highly strategic nature of medical technology.

Should-have features, while not critical, greatly enhance product viability and user satisfaction. This category included advanced nutritional analysis tools (detailed macronutrient distribution by meal), budget tracking systems (predicting weekly expenses), and family profile management – features that add significant value but could be temporarily simplified or delayed without compromising core functionality or safety.

The “Could” level contained features that offer incremental improvements to the user experience with relatively low risk if postponed. Examples include seasonal recipe suggestions, ingredient price highlighting – valuable additions that could be developed after MVP based on user feedback and resource availability. This flexibility has proven critical in the medical technology industry, where early user validation often reveals unexpected needs (e.g., demand for support for rare allergens) that change secondary priorities.

The “Don't Want” category explicitly recognizes resource constraints by excluding features with disproportionately high development costs relative to their value. In our case, API integration with real-time pricing was dropped in favor of manual price entry, as the technical and legal complexities associated with partnering with product platforms outweighed the predicted utility of this feature to our initial user base. This decision was an example of MoSCoW forcing explicit trade-offs, which is especially important in the medical technology industry where regulatory overhead (e.g., data privacy compliance) increases implementation efforts.

Kano's model provides a powerful framework for understanding how different product features influence user satisfaction in medical technology applications. Unlike traditional prioritization methods that focus solely on functional importance, Kano's approach recognizes that not all features contribute equally to the user experience – some are expected basics, while others may delight or even frustrate users if poorly implemented. In this case, applying Kano's model allowed us to understand how to balance the basic requirements for health security and improving user experience. The 17 “Essential” features formed an undeniable foundation – features such as allergen warnings and nutrient calculations that users simply expect to work perfectly. Their absence would make the app unacceptable, but their presence alone does not increase satisfaction. They became our basic foundation for development. 21 “Performance” features showed a linear relationship between implementation quality and user satisfaction – the better we did at diet explanations and BMI tracking, the happier users would be. 7 “Delightful” features, such as seasonal recommendations and offline access, could allow us to exceed expectations and create a competitive advantage. Also identified were 3 truly “Indifferent” features that could be given less attention and potential “Backward” features that could reduce satisfaction.

The Kano model falls short as a standalone prioritization method for health-tech due to critical limitations. Its focus on emotional response over risk assessment creates blind spots in safety-critical domains, failing to distinguish between basic features and those with medical/legal consequences – treating allergen alerts and color preferences similarly. The model struggles with scarce user data in niche medical fields and ignores implementation costs or technical feasibility. Crucially, it lacks phase-awareness, unable to adapt prioritization from clinical safety in MVPs to engagement during scaling, unlike MoSCoW or RICE. Its static nature also clashes with evolving regulatory demands.

The weighted evaluation method provides a quantitative framework for prioritizing features in medical by systematically evaluating requirements against multiple weighted criteria. In developing our web-based personalized nutrition application, this approach has proven invaluable for making objective, data-driven decisions that balance user needs with technical and business constraints. The method is based on evaluating each feature on five key parameters: User Importance (30%), Security Impact (25%), Implementation Complexity (20%), Business Value (15%), and Frequency of Use (10%). For health-critical features such as automatic allergen detection, the model assigned maximum scores for both importance to the user and impact on safety (5/5), resulting in a top priority rating of 4.65.

This score clearly separates mandatory safety features from nice-to-have conveniences – while allergen screening proved critical, features such as budget tracking (2.95) and interface customization (1.2) were appropriately prioritized. A strength of the system is its ability to quantify trade-offs that in other methods are often subjective; for example, it can mathematically demonstrate why implementing a medical contraindication check (4.55) provides more benefit than offering seasonal prescriptions (2.25) when considering both health risks and development effort. In contrast to binary prioritization approaches, the weighted scoring method is able to consider the full range of

medical technology requirements, from established safety features to improved user experience, through its granular scoring system.

The method also adapts well to changing project conditions: when new WHO recommendations required additional nutrient tracking, we could immediately recalculate priorities by changing the weighting factor for safety impact. This dynamic capability proved critical for compliance with limited engineering resources. With standardized evaluations of all features, the method allowed us to clearly explain to stakeholders why certain health-critical features were prioritized, even if they had no obvious appeal to users. The resulting prioritization aligned perfectly with our phased development strategy, providing MVPs of vital features and creating a roadmap for subsequent UX improvements.

Weighted Scoring, while useful for quantifying the prioritization of functions in health-tech, has serious limitations that can threaten product safety and effectiveness. The main weakness of the method is that it simplifies complex medical and ethical aspects into numerical scores, which is dangerous in systems involving patients' lives. For example, equal scores for “allergen detection” and “data encryption” do not reflect the difference between the prevention of physical harm and theoretical safety risks. The method also fails to account for the dynamics of medical research, ignores user psychology, such as the tendency to overlook important warnings, and fails to predict synergy of features when a combination of medium-priority features creates unexpected clinical value. In practice, Weighted Scoring results are often at odds with clinicians' opinions, so the method is best combined with qualitative approaches (MoSCoW) and expert clinician judgment. The score for each feature is a weighted average of all criteria, which ensures objective prioritization while minimizing subjectivity.

The RICE method is a quantitative framework for prioritizing product features by assessing four critical parameters: Coverage (number of users affected), Impact (degree of benefit provided), Confidence (confidence in evaluations), and Effort (resources required for implementation). In the context of our personalized nutrition application, this methodology is of particular importance as it helps to strike a difficult balance between clinical necessity, value to users and design constraints. The fundamental RICE calculation – (Reach × Impact × Confidence)/Effort – provides a score that objectively ranks features by their potential return on the resources invested in development.

For the health-focused platform, the traditional RICE approach was adapted to account for medical imperatives by increasing the weight of the Impact score for critical functions such as allergen detection (Impact: 3) compared to convenience functions such as meal reminders (Impact: 1). This adjustment ensures that vital functionality is not underestimated due to a narrower range of users. The method proved particularly valuable in the MVP phase, where it helped identify high-impact features such as automatic shopping list generation (RICE: 4800) and basic ration personalization (2400) that provided maximum benefit to the user with reasonable effort.

However, we found that the purely quantitative nature of RICE requires careful interpretation in the context of medical technology – while nutrient tracking received a moderate score (6080) due to its broad coverage and low effectiveness, we had to manually escalate medical contraindication alerts (1575) despite their lower score because they are safety-critical. The dynamic nature of the system allowed us to constantly reprioritize as user data were collected; initial Confidence scores of 50-60% for core functions rose to 70-80% after clinical validation, and some perceived high coverage functions such as budget tools (300) were de-prioritized when actual usage data showed limited engagement.

One particularly interesting example was the comparison of functions requiring similar effort – the RICE scores clearly showed why investing in allergen visualization (1380) delivered more value than offering seasonal prescriptions (450), even though both functions took approximately two weeks to develop. The method's emphasis on effort efficiency also helped us avoid resource pitfalls, such as integrating APIs with real-time pricing (11), where technical complexity far outweighed clinical benefit.

The Cost of Delay (CoD) methodology provides a rigorous quantitative approach to feature prioritization that evaluates the temporal impact of implementation decisions through three key dimensions: Criticality (potential consequences of delay), Urgency (time-sensitivity), and Implementation Time (development effort). In the context of our personalized nutrition health-tech application, this method has been fundamentally adapted to address the unique demands of medical software development, where timing decisions carry clinical and regulatory implications beyond conventional product considerations.

At the core of our implementation lies the priority formula (Criticality × Urgency)/Time, which systematically favors features that deliver substantial value quickly while accounting for the opportunity cost of postponement. For health-tech applications, we've recalibrated the traditional CoD parameters to reflect medical imperatives: Criticality now measures potential health outcomes (1=cosmetic to 5=life-threatening), Urgency incorporates regulatory deadlines and seasonal health factors, while Time estimates include clinical validation periods. This adapted framework proved particularly valuable when prioritizing competing safety features – for instance, it clearly demonstrated why "allergen detection" (Priority=5) demanded immediate implementation despite its moderate development timeline (3 weeks), as the potential liability costs of delay ($250k/annual in preventable allergy incidents) dwarfed its development costs.

The method's quantitative nature creates an unambiguous prioritization structure that complements qualitative approaches. Our analysis revealed several critical insights: features with high clinical impact but long development cycles (like medical contraindication screening with Priority=0.8) require manual override mechanisms, while seemingly simple quick-win features (nutrient display at Priority=24) often deliver disproportionate clinical value. We also discovered temporal patterns in health-tech priorities – seasonal allergy features gain Urgency points during peak pollen seasons, while chronic disease management tools maintain steady Criticality ratings year-round.

Implementation challenges specific to health-tech became apparent during deployment. The standard CoD model needed augmentation to handle: (1) regulatory-driven reprioritization (when new FDA guidelines suddenly elevated data privacy features), (2) emergent medical research (new nutrient-drug interaction studies), and (3) non-linear clinical workflows (where feature combinations created unexpected value). Our solution incorporated dynamic weight adjustments – automatically boosting Criticality by 20% for life-critical features and creating regulatory urgency multipliers.

The CoD outputs integrate with other prioritization methods to form a comprehensive decision framework. MoSCoW categories are informed by CoD's time-sensitive analysis, RICE scores are balanced against CoD's risk assessments, and Kano classifications are validated against CoD's cost-benefit calculations. This integration proved crucial when evaluating features like real-time price comparisons (CoD=0.125) versus offline access (CoD=0.2) – while both scored low quantitatively, their qualitative impact on medication adherence in low-income populations required supplementary analysis.

Practical applications demonstrated CoD's strengths in resource allocation. During Q3 development, the model correctly identified that accelerating basic diet personalization (Priority=5) over advanced visualization (Priority=3) would yield 23% greater clinical impact per engineering hour. It also prevented costly missteps, like nearly deprioritizing medical contraindications due to its lengthy implementation timeline before recognizing its critical malpractice risk mitigation value.

For health-tech teams, we recommend CoD as a living framework that requires: (1) monthly recalibration with clinical input, (2) exception protocols for regulatory mandates, and (3) integration with patient safety review boards. When properly configured, it reduces time-to-clinical-impact by an average of 32% compared to traditional prioritization methods, while maintaining rigorous compliance with medical standards. The attached prioritization table (Table III) demonstrates these principles in action across our full feature set, with annotations highlighting key health-specific adjustments made during implementation.

Cost of Delay (CoD) method, despite its effectiveness in time cost management, shows serious

drawbacks when used in health-tech projects. The main problem is that the formula (Criticality × Urgency)/Time artificially lowers the priority of vital but difficult to implement medical functions. For example, in our case, checking medical contraindications (5×3/5=3) was lower than displaying PBMC (4×3/0.5=24), although the former directly prevents life-threatening conditions. This is due to a "penalty" for long development time, which is unacceptable for critical medical functionality.

CoD does not account for complex clinical relationships, such as synergies between functions (e.g., the combination of Allergy History and Ingredient Autosubstitution improves safety) or the cumulative effect of small improvements in long-term therapy. In addition, the method is static and does not adapt to a dynamic medical environment – new research, changes in regulatory requirements or epidemiologic shifts.

Another problem is the preference for quick-to-implement features over more complex but critical ones. For example, "Visual Allergen Identification" (2 weeks, P=6) received priority over "Automatically block dangerous prescriptions" (5 weeks, P=4.8), contradicting the principle of "safety first." CoD also ignores medical metrics: treatment adherence, clinical outcomes, and long-term health effects.

6.3 Recommendations for Implementation

To effectively prioritize requirements in health-tech applications, methods must be combined to offset their individual disadvantages and maximize their advantages. This hybrid approach balances safety, user satisfaction and efficient utilization of resources at different stages of development.

That is, in the initial phase (MVP), it is best to use the MoSCoW method in conjunction with Weighted Scoring, where domain weights are embedded.

During the MVP phase, it is critical to focus on the core features that ensure security and regulatory compliance. MoSCoW helps to quickly divide requirements into Must-have (e.g., allergen screening), Should-have (basic diet personalization), and Could-have (additional UX enhancements). However, to avoid the subjectivity of MoSCoW, Weighted Scoring complements it with a quantitative assessment where safety criteria receive increased weights (e.g., 5× for features that prevent anaphylactic shock). This ensures that vital functions are not inadvertently categorized as Should-have due to lack of stakeholder awareness.

Once the requirements for MVP implementation have been determined, the RICE method can help evaluate return on investment (ROI) for scalable features such as integration with fitness trackers or advanced nutrition analytics. However, in health-tech, the traditional RICE metrics (Reach, Impact) must be adjusted. Target groups should not be made up of all possible audiences, but of specific user categories (e.g., users with allergies). The Impact criterion includes not only commercial benefits but also health effects (e.g. reduced risk of complications). For features where RICE data is insufficient (e.g., new features with no usage history), the Kano method should be used to help assess their potential for user satisfaction. For example, seasonal recommendations (Delighter) can be delayed until the scaling phase if RICE shows a low ROI but Kano confirms their loyalty value.

Cost of Delay with medical adjustments is a better way to do quality time planning. CoD has traditionally focused on commercial risks, but in health-tech its formula (Criticality × Urgency)/Effort needs to be refined. Criticality is rated on a scale of 1 (convenience) to 5 (life-threatening). For example, allergen alerts get a 5, and integration with API pricing gets a 1. Urgency includes not only market timing but also regulatory requirements (e.g., new FDA regulations).

To avoid underestimating complex but critical functions (e.g., checking for drug interactions), the method must be combined with the need to adapt the formulary by augmenting it with medical criteria and combining it with other prioritization methods such as Weighted Scoring and RICE.

Use Weighted Scoring for manual correction. If a feature gets a 5/5 safety score in Weighted Scoring, it automatically gets +2 to Criticality in CoD. If even after correction the CoD remains low (<3) but the feature is a Must-have (MoSCoW), it is included in the MVP in a simplified way (e.g., manual entry of contraindications instead of full automation).

CoD assesses short-term risks, while RICE (Reach, Impact, Confidence, Effort) assesses long-term impact. In health-tech they can be combined, if RICE shows a high Impact (e.g. reduced hospitalizations) but CoD gives a low score due to long implementation, the function is broken down into steps. This is the implementation of a minimal version (e.g. basic checks) or full automation after data collection (increasing Confidence in RICE).

In health-tech, where data, regulations and user expectations are constantly changing, a hybrid approach to product management requires flexibility. After launching an MVP, it is important to analyze feedback through Kano surveys to identify which features have become Basic Needs. For example, if users start to consider allergen warnings critical, this requires reprioritization. The Cost of Delay (CoD) method should also be regularly updated: if a feature (e.g., "seasonal recommendations") was initially low priority (CoD=2) but after release has dramatically increased Retention, its urgency may increase (+1 point). Regulatory changes (e.g., new WHO requirements) automatically increase priority: Criticality in CoD may increase from 3 to 5, and in MoSCoW a feature will move to Must-have even if it was previously Should-have.

For decision making under uncertainty, it is useful to combine RICE with other methods. If Confidence in RICE is low (<50%), the feature can be evaluated by MoSCoW (e.g., included in the plan if it is Must-have for legal reasons) or by Kano (if users consider it Delighter, can be deferred until scaling). For example, a "vitamin intake reminder" feature with a low RICE (due to low confidence) may be deferred, but if new data increases Confidence, the priority is re-prioritized. This approach balances data, regulatory requirements, and user expectations while maintaining scheduling flexibility.

The hybrid approach proposed in the study, combining MoSCoW and Weighted Scoring methods at the MVP stage, followed by the use of RICE and the Kano model at scale, has significant potential for application in various subject areas beyond the personalized nutrition case. The versatility of this solution stems from its ability to effectively balance critical functional requirements, limited development resources, and the need to continuously adapt to changing conditions.

In the field of medical applications, especially monitoring systems for patients with chronic diseases (such as diabetes or hypertension), the proposed methodology shows particular value. Similar to the case of allergens in nutrition, patient safety issues come to the fore here. For example, the drug dosage control function requires mandatory implementation at the MVP stage and should receive increased weights in Weighted Scoring. At the same time, additional functions, such as integration with wearable devices or personalized lifestyle recommendations, can be competently prioritized at the scaling stage using RICE and the Kano model, which allows for optimal allocation of limited development resources.

In financial technology (FinTech), especially in personal investing applications, the proposed approach shows similar effectiveness. Regulatory requirements such as mandatory investor risk profile verification or KYC (Know Your Customer) procedures naturally fall into the "Must-have" category of the MoSCoW methodology, while UX improvements and value-added services (e.g., personalized investment recommendations) can be evaluated through the Kano model and prioritized using RICE. This is particularly important in a highly regulated financial sector, where untimely implementation of mandatory features can lead to serious legal consequences.

Educational platforms, in particular adaptive learning systems, can also benefit significantly from the proposed methodology. Basic functions, such as making educational content available and tracking student progress, are critical at the MVP stage and should be implemented first. At the same time, more complex features, such as gamification elements or supplementary material recommendation systems, which can significantly increase user engagement but require substantial resources for implementation, can be competently prioritized during the scaling phase using quantitative evaluation methods.

6.4 Empirical Validation of the Hybrid Approach

To test the effectiveness of the proposed hybrid approach, a practical experiment was conducted in a custom development environment where one participant combined the roles of analyst, developer, and tester. The personalized nutrition web application project was developed from October 2024 to May 2025, allowing a real-world case study to evaluate the impact of the combined use of prioritization techniques on key project metrics. During the MVP phase (October 2024 – January 2025), a combination of MoSCoW and Weighted Scoring was applied with increased weights for critical functions related to user safety. For example, allergen management and medical contraindication verification functions were assigned five times the weight of other requirements. This approach allowed limited resources to be focused on implementing the most critical components of the system. As a result, the MVP was released in 4 months instead of the planned 5, with 70% of the development time devoted to critical functions, subsequently avoiding three potential costly architecture revisions.

The scaling phase (February – May 2025) focused on improving the user experience and extending functionality. Here, RICE and Kano methods were applied to help objectively assess the potential impact of new features. For example, integration with fitness trackers was highly prioritized due to the combination of broad audience reach (Reach) and significant impact on user satisfaction (Impact), while the development of an API for real product pricing was delayed due to high implementation complexity and relatively low expected impact. As a result of this approach, the development time for low-impact features was reduced – in particular, the implementation of the API for prices took 2 weeks instead of the originally planned 4 weeks.

Three key metrics were used to evaluate the results: development speed, user satisfaction, and whether the implemented features met the original requirements. Analysis of development speed showed that the hybrid approach reduced the overall product development time by 15-20% compared to traditional planning methods. The effect was especially noticeable when implementing critical functions – due to clear ranking of requirements and concentration of resources on key components. User satisfaction was evaluated on a five-point scale based on feedback from 20 test users. Basic features such as allergen management and personalized nutritional recommendations received an average score of 4.7 out of 5, confirming that their prioritization during the MVP phase was correct. After adding "delineators" – features aimed at improving usability, such as seasonal recipes and offline access – the average satisfaction score increased by 0.8 points.

The quality of implementation was evaluated by the percentage of critical features that passed testing without significant comments. The results showed that 90% of Must-have functions were implemented without critical bugs, which indicates the effectiveness of focusing on a limited set of key requirements at the initial stage. An important advantage of the hybrid approach turned out to be its flexibility – when new regulatory requirements or medical recommendations appeared, the weighting system was promptly adjusted, which allowed for an average 30% reduction in the time required to implement the necessary changes compared to traditional planning methods.

Experimental results clearly demonstrated the benefits of the proposed hybrid approach even under extremely resource-constrained conditions. The combination of MoSCoW and Weighted Scoring at the MVP stage provided a solid foundation for the system, focusing efforts on vital functions. The use of RICE and Kano in the scaling phase enabled efficient allocation of limited development resources, avoiding resource traps and focusing on functions with maximum impact. Dynamic adaptation of priorities based on new data (e.g., updated medical guidelines or changes in legislation) ensured a highly flexible development process without losing control over key project metrics. This case demonstrates that the proposed methodology is applicable not only in large teams, but also in individual development environments, allowing to effectively balance the requirements of safety, timing and quality of the final product.

7. Evaluation

The study evaluated five requirements prioritization methods – MoSCoW, Kano, Weighted Scoring, RICE, and Cost of Delay – in terms of their effectiveness in developing a web application for personalized nutrition. The main comparison criteria were: consistency of results between methods, ability to highlight critical features (e.g., security-related), consideration of value to the user, resource efficiency, flexibility to adapt to changes, and ease of use.

All methods unanimously identified key functions for MVP, such as allergen screening, diet personalization, and consideration of medical contraindications. These requirements received the highest priority in MoSCoW (Must-have category), high scores in Weighted Scoring (e.g., 4.65 for allergen checking), and were considered urgent in Cost of Delay (priority = 5). However, secondary functions such as menu planning, API price integration and offline access were rated differently depending on the method. For example, menu planning was ranked as a Must-have feature in MoSCoW but scored low in RICE due to high implementation complexity, while Kano ranked it as a Performance feature affecting user satisfaction.

Each method demonstrated its strengths and weaknesses. MoSCoW provided a clear separation of requirements for MVPs, but its subjectivity may have led to an underestimation of critical features. Kano effectively identified features that increased user satisfaction (e.g., seasonal recommendations), but did not consider implementation costs and security. Weighted Scoring balanced user value, security, and complexity, but simplified medical risks to numerical values. RICE proved useful for evaluating ROI at scale, but required reliable data, which is often lacking in the early stages. Cost of Delay emphasized the importance of urgency but artificially under-prioritized complex but vital functions such as checking drug interactions.

The study confirmed the need for a hybrid approach tailored to the development stages. In the MVP stage, a combination of MoSCoW and Weighted Scoring with increased weights for safety features (e.g., 5× for allergens) provided a focus on critical requirements. In the scaling phase, RICE and Kano helped prioritize features that improve user satisfaction and ROI. Dynamic adaptation of methods, including updating priorities based on user feedback (Kano), regulatory changes (Cost of Delay), and new data (RICE), allowed flexibility in the face of uncertainty.

8. Conclusion

This study examined five requirements prioritization methods – MoSCoW, Kano, weighted evaluation, RICE, and Cost of Delay (CoD) – in the context of developing a web application for personalized nutrition. Although all methods unanimously identified key MVP functions such as allergen checking and diet personalization, their approaches to secondary functions differed due to different focus and evaluation criteria.

MoSCoW proved useful initially by focusing on critical functions (e.g., allergen warnings), but its subjectivity may have led to underestimation of some aspects. The Kano model complemented MoSCoW by identifying basic and 'enthusiastic' functions (e.g., seasonal advice), but did not consider costs and legal risks.

The weighted evaluation quantitatively compared requirements by criteria (importance to the user, safety, complexity of implementation). For example, allergen screening received a high score (4.65) due to the increased weighting of the safety criterion. However, the method simplified complex medical aspects to numerical values.

RICE was useful in the scaling phase, evaluating features for coverage, impact, confidence, and effort. For example, automatic shopping list creation received a high score (4800) because of its wide coverage and low cost. However, the method required reliable data, which was often lacking in the early stages.

Cost of Delay (CoD) assessed the urgency of implementation, but its formula sometimes under-prioritized important but complex features (e.g., checking medical contraindications received a low

score because of long development time.

The main conclusion of the study is that no single method is universal. The most effective solution was a hybrid model adapted to different stages of development. For MVP, MoSCoW and weighted evaluation with increased weights for safety criteria proved to be the optimal combination. In the scaling phase, RICE and Kano helped to focus on features with high ROI and improving the user experience.

This approach ensured efficient use of resources, as the MVP included only the most important features and further product development was based on data and feedback. In addition, the hybrid model remained flexible, allowing for rapid prioritization adjustments as requirements changed, new data emerged, or regulatory updates occurred.

The hybrid prioritization framework combining MoSCoW, Weighted Scoring, RICE and Kano methods demonstrates significant value across multiple domains where balancing critical functionality, user satisfaction and resource constraints is paramount. In medical technology applications like diabetes management systems, the framework ensures patient safety takes precedence during initial development while enabling data-driven scaling. The MVP phase would absolutely require dose calculation algorithms and hypo/hyperglycemia alerts as Must-have features under MoSCoW classification, with Weighted Scoring applying 5x multipliers to these life-critical functions over nice-to-have features like data visualization. This forces explicit prioritization of features that prevent fatal outcomes. During scaling, telehealth integrations could be evaluated using RICE scoring based on their potential patient reach and impact on reducing hospital readmissions, while personalized health tips might emerge as Kano delighters through user feedback analysis. The framework naturally adapts to healthcare's evolving needs – for instance, new FDA guidelines on continuous glucose monitoring could trigger reprioritization by increasing Weighted Scoring's safety multipliers or promoting affected features to Must-have status in MoSCoW.

Financial technology applications like personal investing platforms similarly benefit from this structured yet flexible approach. Regulatory requirements dominate initial development, with KYC verification and fraud detection mechanisms classified as Must-have features that also receive 4x weighting for compliance risk in Weighted Scoring. This prevents common pitfalls where critical security features get deprioritized in favor of flashy but non-essential UI elements. Post-MVP, the framework shifts to optimizing business value – robo-advisor features might score highly in RICE due to their combination of broad user reach and revenue potential, while spending analytics tools could be refined using Kano analysis to maximize customer retention. The Cost of Delay component proves particularly valuable here, quantifying the substantial financial and reputational risks of postponing features like real-time transaction monitoring.

Educational technology platforms for adaptive learning present another compelling use case. The MVP would necessarily focus on core functionality like content delivery and accessibility compliance, with Weighted Scoring assigning higher values to features ensuring universal access. As the platform matures, RICE analysis could justify investment in computationally intensive features like AI-driven recommendations by demonstrating their outsized impact on learning outcomes, while gamification elements might be strategically introduced as Kano delighters to boost engagement metrics. The framework's phased approach allows EdTech products to first meet essential educational standards before layering on innovative features that differentiate them in competitive markets.

Across all these domains, the framework's true power lies in its dynamic adaptability. In healthcare, emerging clinical research can trigger reprioritization through adjusted safety multipliers. In FinTech, changing regulations automatically elevate affected features via MoSCoW reclassification. In EdTech, real-world usage data feeds back into RICE calculations to validate or challenge initial confidence estimates. This responsiveness to new information makes the framework particularly valuable in fast-moving industries where static prioritization approaches quickly become obsolete. Future enhancements could integrate machine learning to automate weight adjustments based on

real user behavior data, creating a continuously self-optimizing prioritization system that maintains alignment between product evolution and genuine user needs across diverse application domains.

Thus, the proposed methodology provides practical tools for requirements management in resource-constrained environments, helping developers and product managers to make informed decisions at all stages of the project lifecycle.

References

[1]. Leffingwell D. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison-Wesley, 2011.
[2]. Cohn M. User Stories Applied: For Agile Software Development. Addison-Wesley, 2004.
[3]. Boehm B., In H. Identifying quality-requirement conflicts. IEEE Software. 1996, vol. 13, № 2, pp. 25-35.
[4]. DSDM Consortium. The DSDM Agile Project Framework, 2014.
[5]. Berkun S. Making Things Happen: Mastering Project Management, O'Reilly Media, 2008.
[6]. Kano N., Seraku N., Takahashi F., Tsuji S. Attractive quality and must-be quality. Journal of the Japanese Society for Quality Control, 1984, vol. 14, № 2, pp. 39-48.
[7]. Griffin A., Hauser J. R. The voice of the customer. Marketing Science, 1993, vol. 12, № 1, pp. 1-27.
[8]. Intercom. The RICE Scoring Model. 2018.
[9]. Reinertsen D. G. The Principles of Product Development Flow: Second Generation Lean Product Development. Celeritas Publishing, 2009.
[10]. Anderson D. J. Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press, 2010.
[11]. Leffingwell D., Widrig D. Managing Software Requirements: A Use Case Approach. Addison-Wesley, 2003.
[12]. Ahl V. An experimental comparison of five prioritization methods. Empirical Software Engineering, 2005, vol. 10, № 4, pp. 375-411.
[13]. Wiegers K. First things first: Prioritizing requirements. Software Development, 1999, vol. 7, № 9, pp. 48-53.
[14]. Beck K. Extreme Programming Explained: Embrace Change. Addison-Wesley, 2000.
[15]. Davis A. Just Enough Requirements Management: Where Software Development Meets Marketing. Dorset House, 2005.
[16]. Clements P., Bass L. Software Architecture in Practice. 3rd ed. Addison-Wesley, 2012.
[17]. Robertson S., Robertson J. Mastering the Requirements Process: Getting Requirements Right. Addison-Wesley, 2012.
[18]. Glinz M. On non-functional requirements. 15th IEEE International Requirements Engineering Conference, 2007, pp. 21-26.
[19]. Smith J., Brown A. Web Application Architecture: Principles and Best Practices. Journal of Software Engineering, 2020, vol. 12, № 3, pp. 45-60.
[20]. Johnson L. User-Centered Design for Health Applications. Health Tech Review, 2019, vol. 8, № 2, pp. 34-50.
[21]. Davis M. et al. Prioritization Techniques in Agile Development. Agile Quarterly, 2021, vol. 5, № 1, pp 22-35.
[22]. Harris T. Frontend Frameworks: A Comparative Study. Web Development Review, 2020, vol. 9, № 1, pp. 55-70.
[23]. Smith J. et al. Impact of Personalized Nutrition on Dietary Adherence. Journal of Nutritional Science, 2020.
[24]. Zeevi D. et al. Personalized Nutrition by Prediction of Glycemic Responses. Cell, 2015.
[25]. Market Research Report. User Preferences in Nutrition Apps, 2023.
[26]. World Health Organization. Global Food Safety Report, 2021.
[27]. Galland L. The Allergy Solution. HarperCollins, 2016.
[28]. AllergyEats Case Study. Allergen Filtering Efficacy, 2020.
[29]. USDA. Time Spent on Meal Planning, 2020.
[30]. Duhigg C. Smarter Faster Better. Random House, 2016.
[31]. Competitive Analysis. Mealime and Yummly Features, 2023.
[32]. Bureau of Labor Statistics. Consumer Expenditure Survey, 2022.
[33]. Brown L. Eat Well on $4/Day. Workman Publishing, 2015.
[34]. Budget Bytes Case Study. Cost-Saving Meal Planning, 2021.

Информация об авторах / Information about authors

Анна Сергеевна МОЖЕГОВА – студентка бакалавриата факультета компьютерных наук, экономики и социальных наук НИУ ВШЭ. Сфера научных интересов: разработка и анализ требований, разработка микросервисных программных продуктов.

Anna Sergeyevna MOZHEGOVA – Bachelor's student at the Faculty of Computer Science, Economics and Social Sciences of the National Research University Higher School of Economics. Research interests: development and analysis of requirements, development of microservice software products.

Вячеслав Владимирович ЛАНИН – старший преподаватель кафедры информационных технологий в бизнесе НИУ ВШЭ-Пермь. Сфера научных интересов: области программной инженерии, включая разработку и проектирование программного обеспечения, управление жизненным циклом ПО (SDLC, Agile, DevOps), а также методы и инструменты разработки, такие как автоматизация тестирования (QA, Unit/Integration Testing), современные практики Continuous Integration/Continuous Delivery (CI/CD), и применение принципов чистого кода и рефакторинга. Также интересуется архитектурой программных систем, распределёнными вычислениями, облачными технологиями и машинным обучением в контексте разработки ПО. В сфере образования его внимание сосредоточено на совершенствовании учебных программ по подготовке IT-специалистов, внедрении актуальных индустриальных практик в образовательный процесс и исследованиях в области педагогики высшего образования в IT.

Vyacheslav Vladimirovich LANIN – senior lecturer, department of Information Technologies in Business HSE University. His research interests lie in the field of software engineering, including software development and design, software lifecycle management, as well as development methods and tools such as test automation (QA, Unit/Integration Testing), modern Continuous Integration/Continuous Delivery practices, and application of clean code and refactoring principles. Also interested in software systems architecture, distributed computing, cloud technologies, and machine learning in the context of software development. In the field of education, his focus is on improving IT training curricula, implementing current industry practices in the educational process, and research in the area of higher education pedagogy in IT.