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.

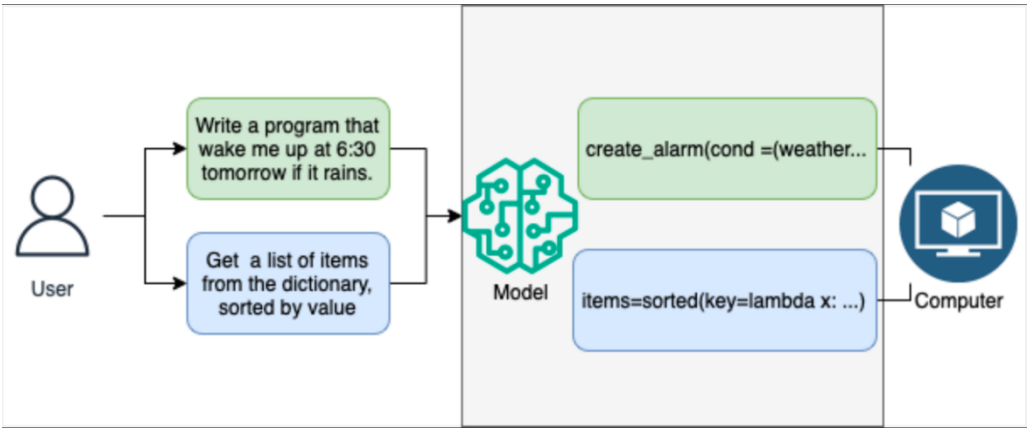

Рис. 1. Сгенерированный код по текстовому описанию зачастую используется без проверок, что может привести к включению в проект или запуску вредоносного кода в системе.

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.