УДК 004.052.42

Формальная верификация программного обеспечения с помощью больших языковых моделей

Мельников Павел Андреевич — магистр, аспирант кафедры «Информатика и вычислительная техника». Научные интересы: надежность программного обеспечения, искусственный интеллект, верификация программного обеспечения. E-mail: gleavero@gmail.com

Тюгашев Андрей Александрович — д-р техн. наук, доцент, профессор кафедры «Информатика и вычислительная техника». Научные интересы: искусственный интеллект, разработка программного обеспечения, надежность программного обеспечения. E-mail: tau797@mail.ru

Институт автоматики и информационных технологий, Самарский государственный технический университет, Россия, 443100, Самарская обл., г. Самара, ул. Молодогвардейская, д. 244

Для цитирования: Мельников П. А., Тюгашев А. А. Формальная верификация программного обеспечения с помощью больших языковых моделей // Интеллектуальные технологии на транспорте. 2025. № 4 (44). С. 47–53. DOI: 10.20295/2413-2527-2025-444-47-53

Аннотация. *Представлено исследование о варианте применения больших языковых моделей для верификации программного обеспечения.* ***Цель:*** *создание системы для автоматической верификации программного обеспечения заданным требованиям. Для достижения цели использованы информационные технологии, формальная верификация, искусственный интеллект и другие инновационные подходы.* ***Методы:*** *анализ современных инструментов и технологий для верификации ПО, включая существующие инструменты.* ***Результаты:*** *показывают сильные и слабые стороны применения больших языковых моделей для верификации ПО.* ***Практическая значимость:*** *заключается в повышении качества и надежности ПО. Исследование имеет важное значение для развития технологий железнодорожного транспорта и повышения надежности работы информационных систем.* ***Обсуждение:*** *высказываются рекомендации по дальнейшему совершенствованию предложенной системы верификации, освещаются вопросы, требующие дальнейших исследований и разработок.*

Ключевые слова: *большие языковые модели, формальная верификация, автоматизация спецификаций, Java Modeling Language, искусственный интеллект*

1.2.1 *— искусственный интеллект и машинное обучение (технические науки);* ***2.3.1*** *— системный анализ, управление и обработка информации (технические науки)*

Введение

Формальные языки спецификации, такие как Java Modeling Language (JML) [1], стали незаменимыми для спецификации и верификации поведения Java-программ. Эти инструменты позволяют разработчикам модульно определять предварительные условия, последующие условия, инварианты и условия кадра, что обычно называется проектированием по контракту. В настоящее время используются дедуктивная верификация (с использованием доказательства теорем и символьного выполнения), ограниченная проверка моделей (на основе теорий выполнимости формул) и новые гибридные методы, включающие машинное обучение для автоматизации генерации спецификаций.

Однако ручное создание аннотаций JML является сложным, трудоемким и подверженным ошибкам процессом. Появление мощных LLM (англ. *Large Language Model* — большая языко-

вая модель, БЯМ) открыло новые возможности для автоматизации генерации спецификаций JML. Интеграция БЯМ с формальными методами может снизить трудоемкость ручной работы и повысить надежность программного обеспечения. Тем не менее остаются следующие проблемы:

- зависимость от контекста — БЯМ могут упускать из виду семантику кода, например побочные эффекты методов;
- синтаксические ошибки — БЯМ могут генерировать код, несовместимый с JML. Модели генерации кода требуют дополнительной настройки, поскольку для точной генерации аннотаций необходимы более объемные обучающие наборы данных;
- семантическая несогласованность — автоматически сгенерированные контракты могут противоречить предполагаемой логике приложения.

Эти проблемы подрывают доверие к автоматизированным решениям и ограничивают их применимость на практике. В результате изучение возможностей интеграции БЯМ с символьной верификацией будет иметь большое значение для снижения рабочей нагрузки при верификации программного обеспечения и повышения качества получаемого программного обеспечения.

Соответственно, цель исследования заключается в изучении точности генерации аннотаций JML с использованием «небольших» языковых моделей (до 10 млрд параметров) и оценке возможности их использования без дополнительной тонкой настройки на примерах кода с существующими аннотациями JML.

Для достижения цели были сформулированы следующие задачи:

- подготовить тестовые примеры для генерации аннотаций JML;
- сгенерировать аннотации и проверить их;
- собрать статистику выполнения и проанализировать качество аннотаций.

Существующие подходы к генерации JML-спецификаций

Традиционно использовались шаблонные методы (например, Houdini) и инструменты вывода инвариантов на основе тестов (например, Daikon). Они эффективно работают для простых спецификаций, но плохо описывают сложную семантику крупных Java-программ.

Новейшие LLM-методики, например SpecGen, используют большие модели для понимания контекста кода и генерации JML-спецификаций. Обычно применяется небольшое количество примеров, и затем спецификации уточняются по результатам проверки; используются мутации и эвристики отбора [2, 3].

Гибридные подходы интегрируют статический анализ и NLP-методы для автоматического извлечения предикатов и обогащения спецификаций (например, проект DECODER): статический анализ дает надежные предикаты, NLP — предикаты из документации [4].

Дедуктивная верификация переводит аннотированные Java-программы в логические формулы; доказательство корректности формул доказывает корректность кода. Инструмент KeY — пример среды для интерактивного доказательства с использованием Java Dynamic Logic [5, 6].

Метод BMC (Bounded Model Checking) исследует пространство состояний до заданного предела, разворачивая циклы и рекурсию фиксированное число раз и проверяя свойства SMT-решателями. Плюс — высокая автоматизация и быстрые контрпримеры, минус — неполнота за пределами границы проверки [7, 8].

LLM применяются для автоматизации создания формальных спецификаций. Мутирование и эвристики отбора в сочетании с верификацией дают улучшенные результаты [9].

Совмещение выводов статического анализа и результатов NLP позволяет получать обогащенные JML-спецификации, итеративная доработка с участием человека повышает качество [4].

Один из новейших подходов улучшает генерацию спецификаций за счет использования операторов мутации для создания разнообразных вариантов — кандидатов. Каждый вариант проверяется, и эвристическая система выбирает тот, который с наибольшей вероятностью является правильным. Такое взаимодействие между БЯМ и формальной верификацией значительно превос-

ходит традиционные статические и основанные на тестах методы, обеспечивая более высокие показатели успешности при валидации [9].

Предлагаемый метод

Предлагаемый метод автоматизации генерации и верификации спецификаций JML состоит из четырех последовательных этапов.

На первом этапе генерируются аннотации. Входным материалом является исходный код классов Java с комментариями на естественном языке. На этом этапе используются локально развернутые модели с открытыми весами: Qwen2.5-coder, CodeLlama и Deepseek-coder-v2 [10, 11].

Сравнение основных параметров моделей приведено в табл. 1. Особое внимание уделяется параметрам (температура 0,7, длина контекста — 4096 токенов), которые позволяют достичь баланса между креативностью и точностью.

На втором этапе проводится синтаксическая проверка. Сгенерированные аннотации проверяются на наличие синтаксических ошибок, типографских несоответствий и конфликтов контрактов с помощью инструмента SpotBugs (версия 4.8.3) [12]. Журналы ошибок записываются для последующего исправления.

На третьем этапе — формальной верификации — спецификации должны быть формально проверены в среде OpenJML [13] (версия 21-0.8) с использованием символьного выполнения и SMT (Satisfiability Modulo Theories)-решателей (логических решателей с поддержкой теорий) (Z3). Неудачные верификации анализируются на предмет таких причин, как недостаточная детализация контракта или семантические несоответствия.

Четвертый этап — классификация и исправление ошибок. Синтаксические ошибки будут автоматически удалены скриптами, а семантические ошибки требуют ручного вмешательства. Результаты записываются в матрицу ошибок, которая служит основой для дальнейшего переобучения моделей.

Экспериментальная установка

Эксперименты проводились на 11 классах Java из примеров кода на веб-сайте OpenJML [14]. Средний размер класса составлял 50–100 строк кода, а функциональность включала методы с циклами, условиями и обработкой исключений. Потенциальным ограничением исследования является относительно небольшой размер выборки, что исключает возможность экстраполяции результатов на более крупные проекты. Для повышения надежности результатов предлагается расширить набор тестовых случаев в последующих работах.

Исследовательская инфраструктура включала MacBook Pro M1 с 16 ГБ унифицированной памяти, который использовался для развертывания LLM и выполнения формальных процедур верификации.

Для оценки производительности использовались следующие метрики:

- точность генерации — доля аннотаций, прошедших синтаксическую верификацию;
- правильность исходного кода — процент скомпилированных классов, прошедших верификацию SpotBugs;
- время обработки класса.

Исходный код экспериментальной установки доступен в репозитории проекта по ссылке http://github.com/Gleavero/verification_system.

Заключение

Основной вклад проведенного исследования заключается в разработке современной инфраструктуры верификации, которая объединяет большие языковые модели с утилитами формальной верификации, такими как SpotBugs и OpenJML. Эта экосистема облегчает оптимизированное создание и подтверждение спецификаций JML.

Предварительные экспериментальные результаты (табл. 2) показывают, что модели умеренного размера, характеризующиеся количеством параметров, не превышающим 7 млрд, демонстрируют способность генерировать синтаксически правильные аннотации.

Однако эффективность этих моделей снижается из-за смешанных переменных, в частности контекстных зависимостей и недостаточно богатого набора данных, используемого на этапе обучения. По различным тестам средняя точность генерации

Таблица 1

Сравнение БЯМ, использованных в работе

	CodeLlama 7B	Qwen-Coder 1.5 (7B)	DeepSeek Coder 6B
HumanEval (Pass@1), %	34,8	41,3	72
Размер контекста, токенов	16 тыс.	8 тыс.	16 тыс.
Размер модели, параметров	7 млрд	7 млрд	6 млрд
Использование памяти, ГБ	14	14	12
Лицензия	Особая (Meta)	Apache 2.0	MIT

Таблица 2

Результаты верификации

Метрика	CodeLlama 7B	Qwen-Coder 1.5 (7B)	DeepSeek Coder 6B
Успешно скомпилировано, %	25,00	58,33	25,00
Прошло через SpotBugs, %	8,33	8,33	8,33
Прошло через OpenJML, %	8,33	8,33	8,33
Среднее время верификации одного класса, с	87,25	34,76	71,12

спецификаций JML составила примерно 8,33 %, хотя были и явные лидеры: Qwen-Coder 1.5 продемонстрировал максимальную производительность на уровне 58 %, а CodeLlama 7B показал промежуточные результаты на уровне 25 %. Анализ протестированных конфигураций выявил несколько общих недостатков. К ним относятся неточности в определении предварительных условий для методов, вызывающих побочные эффекты, и систематическое игнорирование инвариантов на уровне классов (8 %).

По результатам экспериментов предлагаемая методология достигает точности верификации 8,3 %, что ниже стандартов, установленных конкурентами, участвовавшими в конкурсе SV-COMP [15]. Для более высокой точности генерации спецификаций рекомендуется использовать модели, оснащенные расширенными контекстными окнами (минимальная емкость — 16 000 токенов) наряду с архитектурными масштабами, превышающими 10 млрд параметров, типичными для GPT-4 или адаптированных версий CodeLlama. Имеющиеся данные свидетельствуют о том, что использование алгоритмов машинного обучения с синтетическими наборами данных, аннотированными на языке JML, существенно повышает семантическую когерентность получаемых спецификаций.

СПИСОК ИСТОЧНИКОВ

1. Leavens G. T., Cheon Y. Design by Contract with JML. 2006. 13 p. URL: http://www.academia.edu/26405390/Design_by_Contract_with_JML (дата обращения: 07.11.2025).

2. SpecGen: Automated Generation of Formal Program Specifications via Large Language Models / L. Ma, S. Liu, Y. Li // Proceedings of the 47th International Conference on Software Engineering (ICSE 2025) (Ottawa, Canada, 26 April — 06 May 2025). Institute of Electrical and Electronics Engineers, 2025. Pp. 16–28. DOI: 10.1109/ICSE55347.2025.00129.

3. Can Large Language Models Transform Natural Language Intent into Formal Method Postconditions? / M. Endres, S. Fakhoury, S. Chakraborty, S. K. Lahiri // Proceedings of the ACM on Software Engineering. 2024. Vol. 1, Iss. FSE. Art. No. 84. Pp. 1889–1912. DOI: 10.1145/366079.

4. Puccetti A., de Chalendar G., Gibello P.-Y. Combining Formal and Machine Learning Techniques for the Generation of JML Specifications // Proceedings of the 23rd ACM International Workshop on Formal Techniques for Java-like Programs (FTfJP ‘21) (online, 13 July 2021). New York: Association for Computing Machinery. 2021. Pp. 59–64. DOI: 10.1145/3464971.3468425.

5. The Java Verification Tool KeY: A Tutorial / B. Beckert, R. Bubel, D. Drodt [et al.] // Formal Methods (FM 2024): Proceedings of the 26th International Symposium (Milan, Italy, 09–13 September 2024). Part 2. Lecture Notes in Computer Science. Vol. 14934 / A. Platzer [et al.] (eds). Cham: Springer, 2025. Pp. 597–623. DOI: 10.1007/978-3-031-71177-0_32.

6. Hähnle R., Huisman M. Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools // Computing and Software Science: State of the Art and Perspectives. Lecture Notes in Computer Science. Vol. 10000 / B. Steffen, G. Woeginger (eds). Cham: Springer, 2019. Pp. 345–373. DOI: 10.1007/978-3-319-91908-9_18.

7. Modular Verification of JML Contracts Using Bounded Model Checking / B. Beckert, M. Kirsten, J. Klamroth, M. Ulbrich // Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles (ISoLA 2020): Proceedings of the 9th International Symposium on Leveraging Applications of Formal Methods (Rhodes, Greece, 20–30 October 2020). Part 1. Lecture Notes in Computer Science. Vol. 12476 / T. Margaria, B. Steffen (eds). Cham: Springer, 2020. Pp. 60–80. DOI: 10.1007/978-3-030-61362-4_4.

8. Liu T. Efficient Verification of Programs with Complex Data Structures Using SMT Solvers: A Thesis for the Degree of Doctor of Natural Science. Karlsruhe Institute of Technology, 2018. 173 p. DOI: 10.5445/IR/1000084545.

9. SpecEval: Evaluating Code Comprehension in Large Language Models via Program Specifications / L. Ma, S. Liu, L. Bu [et al.] // ArXiv. 2025. Vol. 2409.12866. 12 p. DOI: 10.48550/arXiv.2409.12866.

10. Code Llama: Open Foundation Models for Code / B. Rozière, J. Gehring, F. Gloeckle [et al.] // ArXiv. 2024. Vol. 2308.12950. 48 p. DOI: 10.48550/arXiv.2308.12950.

11. Qwen Technical Report / J. Bai, S. Bai, Y. Chu [et al.] // ArXiv. 2023. Vol. 2309.16609. 59 p. DOI: 10.48550/arXiv.2309.16609.

12. SpotBugs Manual — SpotBugs 4.9.8 Documentation. URL: http://spotbugs.readthedocs.io/en/latest/index.html (дата обращения: 07.11.2025).

13. About OpenJML. URL: http://www.openjml.org/about (дата обращения: 07.11.2025).

14. OpenJML Examples. URL: http://www.openjml.org/examples (дата обращения: 07.11.2025).

15. Beyer D., Strejček J. Improvements in Software Verification and Witness Validation: SV-COMP // Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2025): Proceedings of the 31st International Conference (Hamilton, Canada, 03–08 May 2025). Part 3. Lecture Notes in Computer Science. Vol. 15698 / A. Gurfinkel, M. Heule (eds). Cham: Springer, 2025. Pp. 151–186. DOI: 10.1007/978-3-031-90660-2_9.

Дата поступления: 15.11.2025
Решение о публикации: 18.11.2025

Formal Verification of Software Using Large Language Models

Pavel A. Melnikov — Master of Engineering, Postgraduate Student of the "Computer Science and Computer Engineering" Department. Research interests: software reliability, artificial intelligence, software verification. E-mail: gleavero@gmail.com

Andrey A. Tyugashev — Dr. Sci. in Engineering, Assistant Professor, Professor of the "Computer Science and Computer Engineering" Department. Research interests: artificial intelligence, software development, software reliability. E-mail: tau797@mail.ru

Automation and Information Technology Institute, Samara State Technical University, 244, Molodogvardeyskaya str., Samara, 443100, Russia

For citation: Melnikov P. A., Tyugashev A. A. Formal Verification of Software Using Large Language Models. *Intellectual Technologies on Transport*, 2025, No. 4 (44), Pp. 47–53. DOI: 10.20295/2413-2527-2025-444-47-53. (In Russian)

Abstract. *This study presents an exploration of the application of large language models for software verification.* ***Purpose:*** *to create a system for the automatic verification of software for specified requirements. Information technologies, formal verification, artificial intelligence, and other innovative approaches have been used to achieve this goal.* ***Methods:*** *analysis of current tools and technologies for software verification, including existing instruments.* ***Results:*** *the research highlights the strengths and weaknesses associated with the use of large language models for software verification.* ***Practical significance:*** *enhancing the quality and reliability of software is crucial. This research is important for advancing railway transportation technologies and increasing the reliability of information systems.* ***Discussion:*** *recommendations have been formulated for further improvement of the proposed verification system. Additionally, the issues requiring further research and development have been highlighted.*

Keywords: *large language models, formal verification, specification automation, Java Modeling Language, artificial intelligence*

REFERENCES

1. Leavens G. T., Cheon Y. Design by Contract with JML. 2006. 13 p. Available at: http://www.academia.edu/26405390/Design_by_Contract_with_JML (accessed: November 07, 2025).

2. Ma L., Liu S., Li Y. SpecGen: Automated Generation of Formal Program Specifications via Large Language Models, *Proceedings of the 47th International Conference on Software Engineering (ICSE 2025), Ottawa, Canada, April 26 — May 06, 2025*. Institute of Electrical and Electronics Engineers, 2025, Pp. 16–28. DOI: 10.1109/ICSE55347.2025.00129.

3. Endres M., Fakhoury S., Chakraborty S., Lahiri S. K. Can Large Language Models Transform Natural Language Intent into Formal Method Postconditions? *Proceedings of the ACM on Software Engineering*, 2024, Vol. 1, Iss. FSE, Art. No. 84, Pp. 1889–1912. DOI: 10.1145/366079.

4. Puccetti A., de Chalendar G., Gibello P.-Y. Combining Formal and Machine Learning Techniques for the Generation of JML Specifications, *Proceedings of the 23rd ACM International Workshop on Formal Techniques for Java-like Programs (FTfJP '21), Online, July 13, 2021*. New York, Asso7нн 1ё7ciation for Computing Machinery, 2021, Pp. 59–64. DOI: 10.1145/3464971.3468425.

5. Beckert B., Bubel R., Drodt D., et al. The Java Verification Tool KeY: A Tutorial. In: *Platzer A., et al. (eds) Formal Methods (FM 2024): Proceedings of the 26th International Symposium, Milan, Italy, September 09–13, 2024. Part 2*. Lecture Notes in Computer Science. Vol. 14934. Cham, Springer, 2025, Pp. 597–623. DOI: 10.1007/978-3-031-71177-0_32.

6. Hähnle R., Huisman M. Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools. In: *Steffen B., Woeginger G. (eds) Computing and Software Science: State of the Art and Perspectives.* Lecture Notes in Computer Science. Vol. 10000. Cham, Springer, 2019, Pp. 345–373. DOI: 10.1007/978-3-319-91908-9_18.

7. Beckert B., Kirsten M., Klamroth J., Ulbrich M. Modular Verification of JML Contracts Using Bounded Model Checking, In: *Margaria T., Steffen B. (eds) Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles (ISoLA 2020): Proceedings of the 9th International Symposium on Leveraging Applications of Formal Methods, Rhodes, Greece, October 20–30, 2020. Part 1.* Lecture Notes in Computer Science. Vol. 12476. Cham, Springer, 2020, Pp. 60–80. DOI: 10.1007/978-3-030-61362-4_4.

8. Liu T. Efficient Verification of Programs with Complex Data Structures Using SMT Solvers: A Thesis for the Degree of Doctor of Natural Science. Karlsruhe Institute of Technology, 2018, 173 p. DOI: 10.5445/IR/1000084545.

9. Ma L., Liu S., Bu L., et al. SpecEval: Evaluating Code Comprehension in Large Language Models via Program Specifications, *ArXiv*, 2025, Vol. 2409.12866, 12 p. DOI: 10.48550/arXiv.2409.12866.

10. Rozière B., Gehring J., Gloeckle F., et al. Code Llama: Open Foundation Models for Code, *ArXiv*, 2024, Vol. 2308.12950, 48 p. DOI: 10.48550/arXiv.2308.12950.

11. Bai J., Bai S., Chu Y., et al. Qwen Technical Report, *ArXiv*, 2023, Vol. 2309.16609, 59 p. DOI: 10.48550/arXiv.2309.16609.

12. SpotBugs Manual — SpotBugs 4.9.8 Documentation. Available at: http://spotbugs.readthedocs.io/en/latest/index.html (accessed: November 07, 2025).

13. About OpenJML. Available at: http://www.openjml.org/about (accessed: November 07, 2025).

14. OpenJML Examples. Available at: http://www.openjml.org/examples (accessed: November 07, 2025).

15. Beyer D., Strejček J. Improvements in Software Verification and Witness Validation: SV-COMP. In: *Gurfinkel A., Heule M. (eds) Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2025): Proceedings of the 31st International Conference, Hamilton, Canada, May 03–08, 2025. Part 3*. Lecture Notes in Computer Science. Vol. 15698. Cham, Springer, 2025, Pp. 151–186. DOI: 10.1007/978-3-031-90660-2_9.

Received: 15.11.2025
Accepted: 18.11.2025