К. И. Гладких, А. А. Захаров
Тюменский государственный университет, г. Тюмень
УДК 004.056

ОБЗОР МЕТОДОВ ОБНАРУЖЕНИЯ УЯЗВИМОСТЕЙ В ИСХОДНОМ КОДЕ С ИСПОЛЬЗОВАНИЕМ ИИ-АГЕНТОВ

Аннотация. Ниже мы обозреваем основные выводы из рецензируемых исследований (2019–2025 гг.) по использованию больших языковых моделей, для обнаружения уязвимостей в коде. Мы фокусируемся на методах анализа больших кодовых баз, роли тонкой настройки против поиска, актуальности тонкой настройки языковых моделей, для этой задачи, оценке и сравнения эффективности разных методологии.

Ключевые слова: информационная безопасность, кибербезопасность, большие языковые модели, LLM, анализ уязвимостей, поиск уязвимостей, автоматизация тестирования.

Введение

Актуальность. Количество уязвимостей растет с каждым годом [1], это могут использовать злоумышленники, что приводит к нарушению информационной безопасности систем. Тем самым, преждевременное обнаружение уязвимостей является основной задачей для разработчиков, инженеров по безопасности. Недавние достижения в области искусственного интеллекта (далее ИИ), в частности, большие языковые модели (далее LLM) и другие агенты ИИ, открыли новые возможности для автоматизации и улучшения анализа исходного кода [2]. Эти подходы обладают потенциалом обнаружения сложных уязвимостей, а также предлагают более целостный взгляд на потенциальные риски [3].

Несмотря на растущий интерес к обнаружению уязвимостей с помощью ИИ, литература сильно различается с точки зрения методологии, показателей оценки и применимости к реальному программному обеспечению [3, 4]. Чтобы объединить эти разработки и оценить текущее состояние исследований в этой области, в обзорной статье обозреваются методы обнаружения уязвимостей исходного кода, LLM.

Рамки исследования. В этом обзоре основное внимание уделяется исследованиям, опубликованным с 2019 года, охватывающим статические анализаторы кода на основе языковых моделей. Мы исключаем любые методы, основанные на динамическом анализе, и делаем акцент на общедоступных моделях.

Результаты исследования

RQ1: Какие методы позволяют большим языковым моделям эффективно анализировать и поддерживать необходимый контекст при проверке больших репозиториев на предмет уязвимостей?

Графовые нейросетевые модели: до LLM графовые нейронные модели были популярны в задачах поиска уязвимостей. Например, Devign кодирует исходный код в общий граф (Code/Data Property Graph, далее CDP/DDP) и анализирует представление на уровне графа [5]. Последующие модели интегрировали предварительно обученные вложения кода: ReGVD использует CodeBERT в графовой нейронной сети [6], а GraphCodeBERT интегрирует CDP в трансформерную архитектуру [7].

Генерация с дополненной выборкой (Retrieval-Augmented Generation, далее RAG). Например, Vul-RAG [8] создает базу знаний известных уязвимостей и использует семантический поиск для извлечения паттернов уязвимостей. LLM рассуждает, уязвим ли код, сравнивая его с векторно близкими причинами уязвимости, что увеличило точность. Другой пример, VUL-GPT [9], объединил извлечение фрагментов кода с контекстным обучением GPT-3.

Трансформеры и статические анализаторы кода. Некоторые работы объединяют инструменты статического анализа с LLM для анализа кода всего репозитория. IRIS [10] — это фреймворк, который использует статический анализатор CodeQL для сбора контекстной информации и ее передачи в LLM. Аналогично, подход LSAST [11] совмещает LLM с традиционным статическим анализатором кода.

Выводы. Графовые нейросетевые модели сохраняют структуру кода, тогда как RAG и гибридные методы позволяют LLM общего назначения использовать внешние знания или инструменты. На практике объединение LLM с анализаторами поиска или статическими анализаторами часто дает лучшее покрытие и точность для большого сложного кода, чем использование только LLM.

RQ2: Какие открытые языковые модели используются в задачах обнаружения уязвимостей?

Модели кодировщиков на основе BERT. Имеются исследования, в которых модели на основе BERT были дообучены для обнаружения уязвимостей. Например, была дообучена модель на основе BERT [12] на наборе данных SARD, что позволило достичь точности 93,49%. Аналогичным образом, исследователи дообучили модель RoBERTa на исходном коде [13], чтобы распознавать шаблоны уязвимостей.

CodeBERT — это бимодальный кодирующий трансформатор, обученный на текстовых данных и коде [14]. На бенчмарке CodeXGLUE дообученый CodeBERT достиг точности около 62,1% [15]. GraphCodeBERT дополняет CodeBERT, встраивая графовые представления потоков данных, стремясь захватить зависимости кода [7]. GraphCodeBERT был дообучен методом контрастного обучения с учителем (supervised contrastive learning framework, далее SCL-CVD) для улучшения классификации уязвимого кода [16]. Этот подход, оцененный на наборах данных уязвимостей C/C++. Другой гибридный метод, Vul-LMGNN [17], объединяет предварительно обученный кодировщик CodeBERT с графовой нейронной сетью, оперирующей CPG/DPG.

Модели на базе архитектуры кодировщик-декодировшик. CodeT5 — это модель на основе T5 с 220 млн параметров, обученная на нескольких языках программирования [18]. Она была оценена на задаче обнаружения дефектов CodeXGLUE и достигла точности 65,8%, что существенно превзошло CodeBERT.

Модели на базе архитектуры декодировщика. Другие направления работ используют LLM с архитектурой декодера для статического анализа. Было проведено обширное исследование, сравнение модели семейства GPT с кодерами для обнаружения уязвимостей в коде C/C++ [19]. Результаты показали, что более крупные модели декодера достигли очень высокой точности на бенчмарке VulDeePecker.

Также были оценены недавние общедоступные модели генерации кода. Например, StarCoder и Code LLaMA — это LLM архитектуры декодера, обученные на коде. Дообученая модель может достичь 68,3% в метрике F1 на устаревшем бенчмарке Big-Vul [20] — но только 3,09% F1 на новом бенчмарке PrimeVul.

Статический анализ кода на языке Python. Несколько исследовательских работ были направлены на обнаружение уязвимостей в коде Python, адаптируя модели трансформеров к особенностям Python и его общим уязвимостям. Одним из ярких примеров является DetectVul [21], детектор уязвимостей на уровне операторов, разработанный для Python. DetectVul применяет модель внутреннего внимания на основе BERT непосредственно к функциям Python.

Другой подход, ориентированный на Python, — это использование распознавания именованных сущностей (NER) для улучшения понимания кода. Например, был предложен метод на основе BERT, который токенизирует код Python и помечает токены с помощью методов NER [22].

Эти достижения показывают, что адаптация моделей трансформаторов к синтаксису Python и использование методов обработки естественного языка (например, NER) для кода могут значительно улучшить статическое сканирование уязвимостей в этом языке.

Выводы. Проведенный анализ показывает (табл. 1), что для обнаружения уязвимостей в исходном коде одинаково используются открытые языковые модели трех типов архитектур.

Таблица 1

Сравнение точности обнаружения уязвимостей на языке C/C++

Исследование	*Дообучение*	*CDP/DDP*	*Набор данных*	*Модель*	*Архитектура модели*	*Точность, проценты*
Двоичная классификация						
[16]	✓	✓	SARD	SCL-CVD (Joint Graph Fusion + GAT + Encoder)	Кодировщик	86.50
[14]	-	-	Real-Vul	CodeBERT	Кодировщик (125M)	83.60
[19]	✓	-	VulDeePecker	GPT-2 Large	Декодировщик (774M)	**~96.01**
Многоклассовая классификация						
[12]	✓	-	SARD, NIST NVD	BERT + BiLSTM	Кодировщик (110M)	93.49
[13]	✓	-	Vuldeepecker, Draper, и др.	VulBERTa (RoBERTa)	Кодировщик (125M)	64.75
[17]	✓	✓	Devign	Vul-LMGNN (CodeBERT + GGNN)	Кодировщик (125M)	65.70
			REVEAL			90.80
[18]	-	✓	Devign (CodeXGLUE)	CodeT5 (Wang 2021)	Кодировщик — Декодировщик (220M)	65.78
[19]	✓	-	VulDeePecker	GPT-J	Декодировщик (6B)	**~97.32**

Для Python, пока что сложными остаются бенчмарки Vudenc и CVEFixes (табл. 2). Данные бенчмарки являются актуальными для проверки эффективности языковых моделей в задачах поиска уязвимостей.

Таблица 2

Сравнение точности обнаружения уязвимостей на языке Python и других ЯП

Двоичная классификация						
[20]	✓	-	Big-Vul (SVEN, CodeXGLUE, VulnPatch-Pairs, BigVul, CrossVul, CVEFixes, DiverseVul)	CodeT5	Кодировщик — Декодировщик (60M)	96.33
				CodeBERT	Кодировщик (125M)	96.94
				UnixCoder	Кодировщик (125M)	**97.06**
				StarCoder2	Декодировщик (7B)	97.05
				CodeGen2.5	Декодировщик (7B)	96.94
Многоклассовая классификация						
[23]	✓	-	PythonVulnDB	SecureQwen (CodeQwen1.5)	Декодировщик (7B)	91.67
[22]	✓	-	NVD + Github	CodeBERT	Кодировщик	91.96
				RoBERTa	Кодировщик	91.51
				DistilBERT	Кодировщик	**92.65**
[21]	✓	-	Vudenc	DetectVul (BERT, MiniLM + mpnet)	Кодировщик	66.12
			CVEFixes			82.82

Кодировщики на основе BERT обеспечивают относительно высокую точность, отличаются небольшой размерностью, что облегчает их локальное применение и дообучение. Однако на более сложных задачах и крупных наборах данных их эффективность ниже.

Модели архитектуры кодировщик-декодировщик (CodeT5) демонстрируют заметный прирост производительности по сравнению с чистыми кодировщиками.

Декодировщики, такие как GPT-2 и GPT-J, показывают наивысшую производительность в стандартных наборах данных, однако требуют существенных вычислительных ресурсов для дообучения и использования.

Отдельный анализ потоков данных и исполнения не приносит большого прироста в точности обнаружения уязвимостей на небольших примерах уязвимостей. Исходя из исследований, можно построить архитектуру на базе трансформеров таким образом, чтобы она уже учитывала особенности такие потоков.

Заключение

Недостатки существующих исследований

Преобладание языков C и C++. Языки программирования C и C++ остаются лидирующими, по количеству обнаруженных zero-day уязвимостей и исследований на тему обнаружения уязвимостей. Однако, этот тренд стремительно меняется, по мере того как новые языки программирования набирают популярность, увеличивается количество обнаруженных уязвимостей в исходном коде программ на этих языках [1], например PHP, JavaScript, Python и др.

Недостаток исследований на тему анализа больших кодовых баз языковыми моделями. Большинство исследований сосредоточено на поиске уязвимостей в небольших частях кода, таких как функций, логические блоки и другое. Современные языковые модели способны обрабатывать большое количество текстовой информации, однако в большинстве исследований продолжают использовать устаревшие наборы данных, в которых размеры анализируемого кода намного меньше реальных проектов. Лишь небольшое количество исследований сосредотачивается на анализе потоков исполнения и данных, что является критически важным для понимания природы уязвимостей в крупных и запутанных проектах.

Выводы по ответам на вопросы исследования. В результате проведенного обзора были получены следующие выводы по поставленным вопросам исследования:

- Эффективность анализа больших репозиториев на предмет уязвимостей с помощью больших языковых моделей (LLM) существенно повышается при дополнении анализа контекстными методами. Наибольшие успехи демонстрируют гибридные подходы, такие как использование графовых нейронных сетей, методов генерации с дополненной выборкой и интеграции с традиционными статическими анализаторами.
- Среди открытых моделей языков, используемых в задачах обнаружения уязвимостей, наибольшую популярность приобрели модели на основе архитектуры BERT. Модели на основе декодировщиков демонстрируют высокую точность.

Несмотря на достигнутые успехи, существуют существенные ограничения текущих исследований. Большинство исследований фокусируются на языках программирования C и C++, в то время как другие популярные языки (например, Python, JavaScript) менее изучены. Кроме того, существующие исследования редко учитывают сложности анализа реальных крупных проектов с многочисленными потоками данных и исполнения.

Направления для будущих исследований. Перспективными направлениями для будущих исследований являются:

- Анализ эффективности обнаружения уязвимостей в других языках программирования отличных от C и C++.
- Разработка масштабируемых подходов, способных обрабатывать крупные кодовые репозитории целиком, включая потоки исполнения и потоки данных.
- Улучшение качества и репрезентативности наборов данных для обучения и оценки LLM, в том числе путем создания и использования новых, более реалистичных и разнообразных наборов данных (бенчмарков).
- Исследование механизмов интеграции LLM с другими типами анализа (например, динамическим и поведенческим анализом) для более полного охвата потенциальных уязвимостей.

СПИСОК ЛИТЕРАТУРЫ

1. Gladkikh K.I. Characteristics and Trends of Zero-Day Vulnerabilities in Open-Source Code / A. A. Zakharov, K. I. Gladkikh — Текст : непосредственный // International Russian Automation Conference (RusAutoCon). 8-14 сен. 2024 г. — Сочи, 2024.
2. Дроздов В. А. Применение больших языковых моделей для анализа уязвимостей / В. А. Дроздов // Научный аспект. — 2024. — Т. 45, № 6. — С. 5631-5637. — EDN NUYNZP.

3. Можаровский Е. А. Применение больших языковых моделей для автоматизации тестирования и отладки программного обеспечения / Е. А. Можаровский, А. С. Алуев, А. А. Дудак, А. Ю. Ишанхонов // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и технические науки. — 2024. — № 11-2. — С. 103-108. — DOI 10.37882/2223-2966.2024.11-2.22. — EDN YSUWDL.
4. Котенко И. В. Использование больших языковых моделей для поиска угроз кибербезопасности на основе методов глубокого обучения: анализ современных исследований / И. В. Котенко, Г. Т. Абраменко // Правовая информатика. — 2024. — № 3. — С. 32-42. — DOI 10.24682/1994-1404-2024-3-32-42. — EDN SBQLOI.
5. Zhou Y. Devign: effective vulnerability identification by learning comprehensive program semantics via graph neural networks / Y. Zhou, S. Liu, J. Siow, X. Du, Y. Liu. — Текст : непосредственный // Proceedings of the 33rd International Conference on Neural Information Processing Systems. — Red Hook, NY, USA : Curran Associates Inc., 2019. — Art. 915. — С. 10197–10207.
6. Nguyen V. ReGVD: revisiting graph neural networks for vulnerability detection / V. Nguyen, D. Q. Nguyen, V. Nguyen, T. Le, Q. H. Tran, D. Phung. — Текст : непосредственный ; электронный // Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings (ICSE '22). — New York, NY, USA : Association for Computing Machinery, 2022. — С. 178–182. — DOI: 10.1145/3510454.3516865
7. Guo D. Graphcodebert: Pre-training code representations with data flow / D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano. — Текст : непосредственный ; электронный // arXiv preprint. — 2020. — № arXiv:2009.08366. — URL: https://arxiv.org/abs/2009.08366 (дата обращения: 15.04.2025).
8. Du X. Vul-rag: Enhancing llm-based vulnerability detection via knowledge-level rag / X. Du, G. Zheng, K. Wang, J. Feng, W. Deng, M. Liu, B. Chen, X. Peng, T. Ma, Y. Lou. — Текст : непосредственный ; электронный // arXiv preprint. — 2024. — № arXiv:2406.11147. — URL: https://arxiv.org/abs/2406.11147 (дата обращения: 15.04.2025).
9. Liu Z. Software vulnerability detection with gpt and in-context learning / Z. Liu, Q. Liao, W. Gu, C. Gao. — Текст : непосредственный // 2023 8th International Conference on Data Science in Cyberspace (DSC) (18 августа 2023 г.). — IEEE, 2023. — С. 229–236.
10. Li Z. IRIS: LLM-assisted static analysis for detecting security vulnerabilities / Z. Li, S. Dutta, M. Naik. — Текст : непосредственный // The Thirteenth International Conference on Learning Representations. — 2025.
11. Keltek M. LSAST–Enhancing Cybersecurity through LLM-supported Static Application Security Testing / M. Keltek, R. Hu, M. F. Sani, Z. Li. — Текст : непосредственный ; электронный // arXiv preprint. — 2024. — № arXiv:2409.15735. — URL: https://arxiv.org/abs/2409.15735 (дата обращения: 15.04.2025).
12. Ziems N. Security vulnerability detection using deep learning natural language processing / N. Ziems, S. Wu. — Текст : непосредственный // IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (10 мая 2021 г.). — IEEE, 2021. — С. 1–6.
13. Hanif H. Vulberta: Simplified source code pre-training for vulnerability detection / H. Hanif, S. Maffeis. — Текст : непосредственный // 2022 International Joint Conference on Neural Networks (IJCNN) (18 июля 2022 г.). — IEEE, 2022. — С. 1–8.
14. Feng Z. Codebert: A pre-trained model for programming and natural languages / Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, M. Zhou. — Текст : непосредственный ; электронный // arXiv preprint. — 2020. — № arXiv:2002.08155. — URL: https://arxiv.org/abs/2002.08155 (дата обращения: 15.04.2025).
15. Lu S. Codexglue: A machine learning benchmark dataset for code understanding and generation / S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang, G. Li. —

Текст : непосредственный ; электронный // arXiv preprint. — 2021. — № arXiv:2102.04664. — URL: https://arxiv.org/abs/2102.04664 (дата обращения: 15.04.2025).
16. Jin D. Source Code Vulnerability Detection Based on Joint Graph and Multimodal Feature Fusion / D. Jin, C. He, Q. Zou, Y. Qin, B. Wang. — Текст : непосредственный // Electronics. — 2025. — Т. 14, № 5. — С. 975.
17. Liu R. Source code vulnerability detection: Combining code language models and code property graphs / R. Liu, Y. Wang, H. Xu, B. Liu, J. Sun, Z. Guo, W. Ma. — Текст : непосредственный ; электронный // arXiv preprint. — 2024. — № arXiv:2404.14719. — URL: https://arxiv.org/abs/2404.14719 (дата обращения: 15.04.2025).
18. Wang Y. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation / Y. Wang, W. Wang, S. Joty, S. C. Hoi. — Текст : непосредственный ; электронный // arXiv preprint. — 2021. — № arXiv:2109.00859. — URL: https://arxiv.org/abs/2109.00859 (дата обращения: 15.04.2025).
19. Thapa C. Transformer-based language models for software vulnerability detection / C. Thapa, S. I. Jang, M. E. Ahmed, S. Camtepe, J. Pieprzyk, S. Nepal. — Текст : непосредственный // Proceedings of the 38th Annual Computer Security Applications Conference (5 декабря 2022 г.). — 2022. — С. 481–496.
20. Ding Y. Vulnerability detection with code language models: How far are we? / Y. Ding, Y. Fu, O. Ibrahim, C. Sitawarin, X. Chen, B. Alomair, D. Wagner, B. Ray, Y. Chen. — Текст : непосредственный ; электронный // arXiv preprint. — 2024. — № arXiv:2403.18624. — URL: https://arxiv.org/abs/2403.18624 (дата обращения: 15.04.2025).
21. Tran H. C. DetectVul: A statement-level code vulnerability detection for Python / H. C. Tran, A. D. Tran, K. H. Le. — Текст : непосредственный // Future Generation Computer Systems. — 2025. — Т. 163. — С. 107504.
22. Ehrenberg M. Python source code vulnerability detection with named entity recognition / M. Ehrenberg, S. Sarkani, T. A. Mazzuchi. — Текст : непосредственный // Computers & Security. — 2024. — Т. 140. — С. 103802.