ТЕХНИКАЛЫҚ ҒЫЛЫМДАР
ТЕХНИЧЕСКИЕ НАУКИ
TECHNICAL SCIENCE

CENTRAL ASIAN SCIENTIFIC JOURNAL
ISSN 3135-3061
2026. №4 (32), Том 2. С. 582–586
DOI: https://doi.org/10.5281/zenodo.24458260
Сайт журнала: https://cajournal.kz

УДК 004.056.5:004.8

AI-СИСТЕМЫ ОБНАРУЖЕНИЯ УЯЗВИМОСТЕЙ ИСХОДНОГО КОДА

А.А. Устинов, С.А. Томашевский
Satbayev University, Алматы, Казахстан

Устинов Артём Александрович — студент 4 курса образовательной программы «Информационная безопасность», Satbayev University, Алматы, Казахстан.

Томашевский Сергей Андреевич — студент 4 курса образовательной программы «Информационная безопасность», Satbayev University, Алматы, Казахстан.

© А.А. Устинов, С.А. Томашевский, 2026

Аннотация: В статье представлен обзор современных подходов к автоматическому обнаружению уязвимостей исходного кода с использованием методов искусственного интеллекта. Рассмотрены три поколения AI-систем: классические методы машинного обучения, модели глубокого обучения на основе графовых нейронных сетей и предобученных трансформеров, а также большие языковые модели. Проведён сравнительный анализ точности моделей на эталонных наборах данных. Показано, что гибридные подходы, сочетающие статический анализ и нейросетевые модели, обеспечивают наилучшие результаты обнаружения при допустимом уровне ложных срабатываний.

Ключевые слова: обнаружение уязвимостей, искусственный интеллект, глубокое обучение, большие языковые модели, статический анализ кода, безопасность программного обеспечения, нейронные сети

Введение

Безопасность программного обеспечения является одной из ключевых проблем современной информатики. По данным базы MITRE CVE, ежегодно регистрируется более 25 000 новых уязвимостей, причём их количество демонстрирует устойчивый рост [1]. Уязвимости в исходном коде, такие как переполнение буфера (CWE-119), запись за пределы буфера (CWE-787) и использование памяти после освобождения (CWE-416), входят в список 25 наиболее опасных слабостей программного обеспечения CWE Top 25 и ежегодно становятся причиной масштабных инцидентов информационной безопасности [2].

Традиционные средства статического анализа кода (SAST), такие как Semgrep, SonarQube и Snyk, основаны на детерминированных правилах и шаблонах. Они обеспечивают низкий уровень ложных срабатываний, однако обладают ограниченной способностью к обнаружению сложных, контекстно-зависимых уязвимостей [3]. Появление методов искусственного интеллекта и, в частности, глубокого обучения открыло новые возможности для автоматизации поиска дефектов безопасности в коде.

Цель настоящей работы - провести систематический обзор AI-систем обнаружения уязвимостей исходного кода, классифицировать существующие подходы по типу используемых моделей, сравнить их эффективность на стандартных наборах данных и определить перспективные направления развития данной области.

Материалы и методы

Для проведения обзора использовался метод систематического анализа литературы. Поиск публикаций проводился в базах данных IEEE Xplore, ACM Digital Library, Scopus и arXiv по ключевым словам: «vulnerability detection», «deep learning», «source code analysis», «LLM security». Критерием включения являлось наличие экспериментальной оценки на стандартных наборах данных. Период охвата -2018-2026 гг. Всего было отобрано и проанализировано 15 ключевых работ.

Классификация подходов проведена по типу используемой модели: (1) классические методы машинного обучения; (2) модели глубокого обучения на основе рекуррентных и графовых нейронных сетей; (3) предобученные модели-трансформеры; (4) большие языковые модели (LLM). Для сравнительного анализа использовалась метрика F1-score как наиболее информативная для задач с дисбалансом классов, характерным для наборов данных по уязвимостям.

Основными эталонными наборами данных в рассмотренных работах являются:

- Devign - набор вручную размеченных уязвимых и безопасных функций из четырёх крупных проектов на языке C [4];

- Big-Vul - набор изменений кода на C/C++, связанных с записями CVE за период 2002-2019 гг. [5];

- CVEfixes - автоматически собранный набор уязвимых функций из 564 проектов с открытым исходным кодом, содержащий 8 932 уязвимые функции [6];

- MegaVul - расширенный набор данных для C/C++, в котором уязвимости, связанные с безопасностью памяти (CWE-119, CWE-125, CWE-787), составляют 59,27 % [7].

Результаты и обсуждение

На основе проведённого анализа были выделены четыре поколения AI-систем обнаружения уязвимостей, отражающие эволюцию подходов к представлению кода и архитектуре моделей.

Классические методы машинного обучения. Ранние работы использовали ручное извлечение признаков из исходного кода (метрики сложности, частота вызовов опасных функций, статистические характеристики потока управления) с последующим обучением классификаторов, таких как Random Forest и SVM. Основным ограничением данного подхода является зависимость от качества экспертно выбранных признаков и неспособность моделей улавливать глубокие семантические зависимости в коде [8].

Модели на основе рекуррентных и графовых нейронных сетей. Переход к глубокому обучению позволил автоматически извлекать представления кода. Li и др. предложили систему VulDeePecker, использующую двунаправленную LSTM для анализа последовательностей токенов кода и достигшую значимого улучшения по сравнению с традиционными методами [9]. Дальнейшее развитие связано с использованием графовых нейронных сетей (GNN), работающих с абстрактным синтаксическим деревом (AST) и графом потока данных. Модель Devign, основанная на GNN, продемонстрировала способность учитывать структурные связи между элементами кода [4].

Предобученные трансформерные модели. Значительный прорыв связан с адаптацией предобученных моделей-трансформеров к задачам анализа кода. CodeBERT -мультиязычная модель, обученная на парах «код - документация» из шести языков программирования, - стала базовой моделью для многих последующих работ [10]. GraphCodeBERT дополняет текстовое представление кода информацией о потоке данных, что позволяет повысить точность обнаружения уязвимостей [11]. LineVul использует архитектуру трансформера с механизмом внимания на уровне строк кода для локализации уязвимых фрагментов [12].

Результаты сравнительного анализа эффективности предобученных моделей на эталонных наборах данных представлены в таблице 1.

Таблица 1 - Сравнение F1-score моделей на эталонных наборах данных

Модель	Devign	CVEfixes	Big-Vul

CodeBERT	0,573	0,603	0,412
GraphCodeBERT	0,645	0,603	0,458
LineVul	0,638	0,586	0,441
CodeT5	0,671	0,590	0,453

Визуальное сравнение F1-score моделей по наборам данных представлено на рисунке 1.

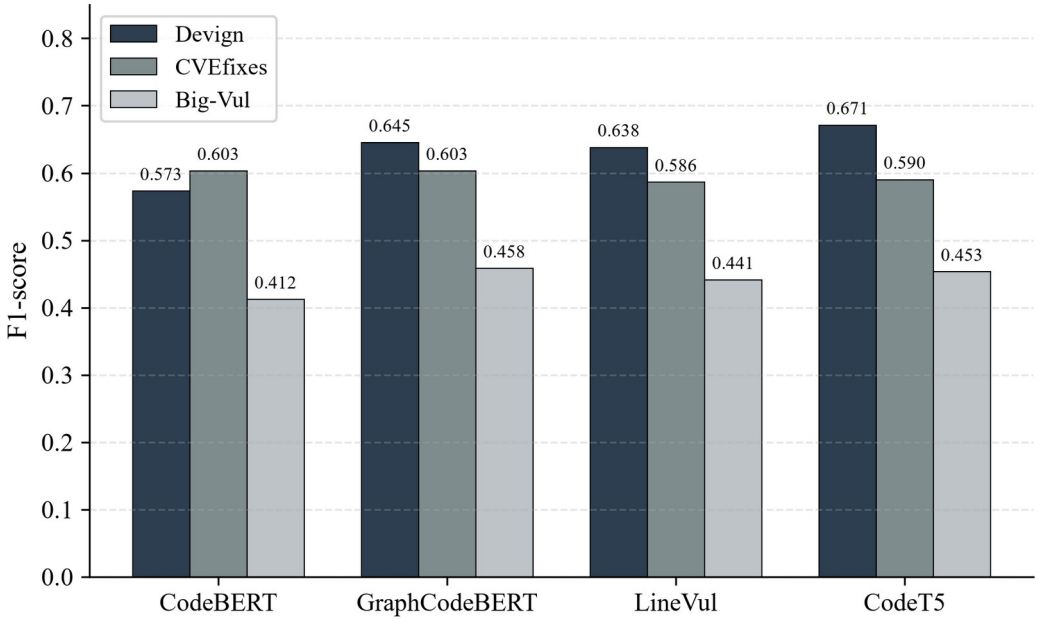


Рисунок 1 - Сравнение F1-score предобученных моделей на эталонных наборах данных

Данные таблицы 1 получены из результатов бенчмарка VulnBench [13]. Следует отметить, что значения F1-score существенно различаются в зависимости от набора данных. На синтетических наборах (Juliet Test Suite) модели демонстрируют F1 = 0,900, тогда как на реальных данных (DiverseVul) - лишь 0,307, что свидетельствует о значительном разрыве между лабораторными и практическими условиями (рисунок 2) [13].

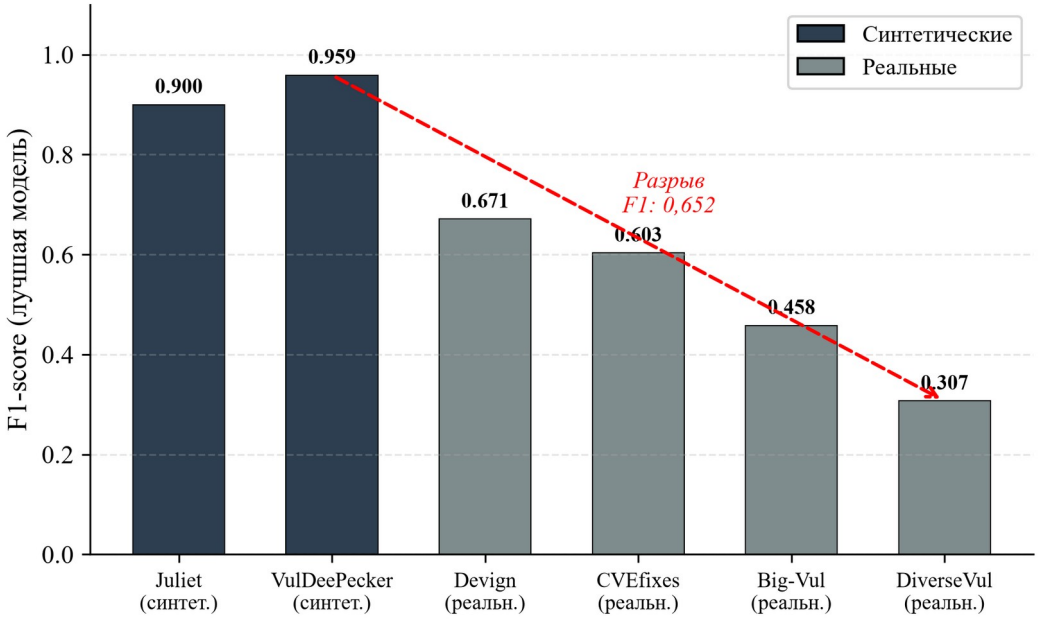


Рисунок 2 - Разрыв F1-score между синтетическими и реальными наборами данных

Большие языковые модели. Новейшим направлением является применение LLM, таких как GPT-4, Claude и специализированных моделей (StarCoder, CodeLlama). Исследование EMNLP 2025 показало, что среди LLM-подходов GPT-4o демонстрирует наилучшие результаты, однако предложенная авторами многоступенчатая архитектура

превосходит GPT-4o на 6,5 % по F1-score [14]. Преимуществом LLM является способность к обнаружению уязвимостей в режиме «zero-shot» без дополнительного обучения, однако они сопряжены с высоким уровнем ложных срабатываний и значительными вычислительными затратами.

Гибридные подходы. Наиболее перспективным направлением является сочетание детерминированного статического анализа и нейросетевых моделей. Система IRIS, представленная на ICLR 2025, комбинирует LLM с анализатором CodeQL и превосходит CodeQL, используемый отдельно [3]. SAST-Genius (IEEE S&P 2025) использует LLM для фильтрации ложных срабатываний традиционных SAST-инструментов [3]. Бенчмарк RealVuln, проведённый на 26 уязвимых Python-репозиториях, показал, что гибридные LLM-сканеры набирают F3-score почти втрое выше, чем Semgrep, применяемый автономно [15].

Результаты сравнения подходов к обнаружению уязвимостей по ключевым характеристикам представлены в таблице 2.

Таблица 2 - Сравнительная характеристика подходов к обнаружению уязвимостей

Подход	Полнота	Точность	Скорость	Объяснимость
SAST (правила)	Низкая	Высокая	Высокая	Высокая
ML (признаки)	Средняя	Средняя	Высокая	Средняя
DL (трансформеры)	Высокая	Средняя	Средняя	Низкая
LLM (zero-shot)	Высокая	Низкая	Низкая	Средняя

Анализ выявил ряд ключевых проблем, ограничивающих практическое применение AI-систем обнаружения уязвимостей.

Во-первых, качество эталонных наборов данных остаётся серьёзной проблемой. Исследование показало, что 45,7 % меток в Big-Vul и 20 % меток в Devign содержат ошибки разметки, что ставит под вопрос надёжность результатов, полученных на этих данных [13]. Кроме того, наборы данных по безопасности характеризуются сильным дисбалансом классов: доля уязвимого кода обычно не превышает 3-5 % от общего объёма, что приводит к завышенным показателям accuracy при низкой реальной полноте обнаружения [13].

Во-вторых, обобщающая способность моделей остаётся ограниченной. Модели, обученные на одном наборе данных, показывают значительное падение производительности при переносе на другой. Так, модели, демонстрирующие F1 = 0,900 на синтетическом наборе Juliet, достигают лишь F1 = 0,307 на реальных данных DiverseVul [13]. Эмпирическая оценка на вновь обнаруженных уязвимостях CVE 2024 года показала, что ни один из существующих детекторов не обеспечил удовлетворительных результатов [16].

В-третьих, объяснимость решений нейросетевых моделей недостаточна для практического использования в рабочих процессах разработки. Разработчики нуждаются не только в указании на потенциально уязвимую строку кода, но и в обосновании того, почему данный фрагмент является уязвимым и как его исправить. Механизмы внимания (attention) трансформерных моделей обеспечивают частичную интерпретируемость, но не заменяют полноценного объяснения [12].

Заключение

Проведённый обзор показал, что AI-системы обнаружения уязвимостей исходного кода прошли путь от классических методов машинного обучения с ручным извлечением признаков до предобученных трансформерных моделей и больших языковых моделей. Каждое поколение расширяет спектр обнаруживаемых уязвимостей, однако ни один из существующих подходов не решает задачу полностью.

Предобученные трансформерные модели (CodeBERT, GraphCodeBERT, CodeT5) обеспечивают наилучший баланс между полнотой и точностью обнаружения. Большие языковые модели демонстрируют способность к обнаружению уязвимостей без дополнительного обучения, но сопряжены с высоким уровнем ложных срабатываний.

Гибридные подходы, комбинирующие статический анализ и нейросетевые модели, представляют наиболее перспективное направление.

Дальнейшее развитие данной области требует: (1) создания качественных наборов данных с верифицированной разметкой; (2) разработки методов повышения обобщающей способности моделей на реальном коде; (3) интеграции механизмов объяснимости в AI-детекторы уязвимостей; (4) создания стандартизированных бенчмарков для воспроизводимого сравнения подходов.

Список литературы:

1. CWE - 2025 CWE Top 25 Most Dangerous Software Weaknesses [Электронный ресурс]. - URL: https://cwe.mitre.org/top25/ (дата обращения: 15.11.2026).

2. MITRE. Common Weakness Enumeration (CWE) [Электронный ресурс]. - URL: https://cwe.mitre.org/ (дата обращения: 15.11.2026).

3. Comparison of Static Application Security Testing Tools and Large Language Models for Repo-level Vulnerability Detection / M. Wen, Y. Chen, D. Lu [et al.] // arXiv preprint. - 2024. - arXiv:2407.16235.

4. Zhou Y., Liu S., Siow J. et al. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks // Advances in Neural Information Processing Systems (NeurIPS). - 2019. - Vol. 32. - P. 10197-10207.

5. Fan J., Li Y., Wang S. et al. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries // Proceedings of the 17th International Conference on Mining Software Repositories (MSR). - 2020. - P. 508-512.

6. Bhandari G.P., 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 (PROMISE). - 2021. - P. 30-39.

7. Ni C., Yin K., Yang S. et al. MegaVul: A C/C++ Vulnerability Dataset with Comprehensive Code Representation // arXiv preprint. - 2024. - arXiv:2406.12415.

8. Survey of source code vulnerability analysis based on deep learning // Computers & Security. - 2025. - Vol. 148. - Art. 104098.

9. Li Z., Zou D., Xu S. et al. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection // Proceedings of the Network and Distributed System Security Symposium (NDSS). - 2018.

10. Feng Z., Guo D., Tang D. et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages // Findings of the Association for Computational Linguistics: EMNLP 2020. - P. 1536-1547.

11. Guo D., Ren S., Lu S. et al. GraphCodeBERT: Pre-training Code Representations with Data Flow // Proceedings of the International Conference on Learning Representations (ICLR). - 2021.

12. Fu M., Tantithamthavorn C. LineVul: A Transformer-based Line-Level Vulnerability Prediction // Proceedings of the 19th International Conference on Mining Software Repositories (MSR). - 2022. - P. 608-620.

13. VulnBench: A Comprehensive Benchmark for Transformer-Based Vulnerability Detection // Proceedings of the AAAI Conference on Artificial Intelligence. - 2025.

14. A Sequential Multi-Stage Approach for Code Vulnerability Detection // Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP). - 2025.

15. RealVuln: Benchmarking Rule-Based, General-Purpose LLM, and Security-Specialized Scanners on Real-World Code // arXiv preprint. - 2026. - arXiv:2604.13764.

16. It Only Gets Worse: Revisiting DL-Based Vulnerability Detectors from a Practical Perspective // arXiv preprint. - 2025. - arXiv:2507.09529.