УДК 004.415.5:004.8

Бадертдинова Елена Радитовна, доктор технических наук, ФГБОУ ВО «Казанский национальный исследовательский технологический университет», г. Казань

Совгачев Александр Андреевич, магистрант, ФГБОУ ВО «Казанский национальный исследовательский технологический университет», г. Казань

АВТОМАТИЧЕСКАЯ ГЕНЕРАЦИЯ ТЕСТОВ СРЕДСТВАМИ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ: КАЧЕСТВО, ОГРАНИЧЕНИЯ И ФЕНОМЕН ЛОЖНОЙ БЕЗОПАСНОСТИ

Аннотация

Статья представляет обзор эмпирических исследований 2021–2026 годов, посвящённых автоматической генерации тестов с помощью больших языковых моделей (LLM). Рассматривается эволюция подходов: от базовой промпт-генерации до фреймворков с repair-loop (TestPilot, ChatUniTest, ChatTESTER); анализируются метрики качества: покрытие кода, корректность ассерций, mutation score. Показано, что LLM превосходят традиционные SBST-инструменты по читаемости, однако уступают в обнаружении дефектов (mutation score ~18,9%). Выявлены систематические структурные дефекты: Assertion Roulette (>50% тестов при zero-shot генерации), нестабильность (63% случаев из-за неупорядоченных коллекций), избыточное мокирование AI-агентами. По итогам синтеза данных авторами предложена концепция «ложной безопасности» (иллюзии достаточного тестирования при структурно дефектных тестах) с классификацией пяти механизмов её формирования.

Annotation

This paper presents a review of empirical research from 2021–2026 on automated test generation using large language models (LLMs). We trace the evolution of approaches from basic prompt generation to repair-loop frameworks

(TestPilot, ChatUniTest, ChatTESTER) and analyze quality metrics: code coverage, assertion correctness, and mutation score. LLMs produce more readable tests than search-based tools (EvoSuite) but lag in fault detection (mutation score ~18.9%). Systematic structural defects are identified: Assertion Roulette (>50% of zero-shot tests), flakiness (63% due to unordered collections), and over-mocking by AI agents. Based on empirical synthesis, we propose the concept of "false confidence" (the illusion of adequate testing coverage produced by structurally deficient tests) and classify five mechanisms through which it forms.

Ключевые слова: большие языковые модели, автоматическая генерация тестов, unit-тестирование, покрытие кода, mutation score, test smells, ложная безопасность, EvoSuite, ChatGPT, качество программного обеспечения.

Keywords: large language models, automated test generation, unit testing, code coverage, mutation score, test smells, false confidence, EvoSuite, ChatGPT, software quality.

Введение

По отраслевым оценкам начала 2000-х годов, от 40 до 50% ресурсов проекта уходит на верификацию и тестирование [11]; последующие работы подтверждают, что тестирование занимает значительную долю совокупных затрат на разработку. Автоматизация написания тестов остаётся нерешённой задачей программной инженерии. Традиционный ответ на неё — поисковое тестирование, реализованное, в частности, в инструменте EvoSuite для Java [6], и методы фаззинга. Эти подходы позволяют максимизировать покрытие кода, однако порождают тесты с криптическими идентификаторами, нечитаемыми ассерциями и отсутствием смысловой структуры.

Предобученные языковые модели открыли совершенно иное направление: генерацию тестов в виде читаемого, семантически осмысленного кода непосредственно по исходному тексту тестируемого класса или функции. Исследовательский интерес к теме нарастал стремительно. По данным обзора Чжу и соавт., охватывающего 115 публикаций за 2021–2025 годы, в 2024 году вышло 73 работы по теме — против 14 в 2023-м [4]. Промышленность

отреагировала сопоставимо быстро: инструменты на базе LLM встроены в GitHub Copilot, JetBrains AI Assistant, Cursor и ряд специализированных платформ.

Однако за стремительным ростом числа инструментов и публикаций теряется вопрос, который формулируется значительно реже: можно ли доверять сгенерированным тестам? Присутствие тестов в репозитории и ненулевое покрытие кода формируют у команды субъективное ощущение надёжности системы. Если тесты структурно дефектны — страдают от test smells, содержат тривиальные ассерции или изолируют зависимости вместо их проверки — это ощущение превращается в то, что авторы настоящей работы предлагают обозначить как «ложную безопасность»: иллюзию достаточного тестирования, порождаемую метриками покрытия при структурно дефектных тестах.

Цель настоящей статьи — синтезировать эмпирические данные 2021–2026 годов о качестве LLM-генерируемых тестов, концептуализировать механизмы ложной безопасности и обозначить практические следствия для команд разработки.

Методология обзора

Поиск источников проводился в базах arXiv (cs.SE), ACM Digital Library, Springer Link, CyberLeninka по запросам «LLM unit test generation», «automated test generation large language model», «test smells LLM», «LLM flaky tests», «генерация тестов языковые модели». Критерии включения: наличие эмпирических данных о качестве LLM-тестов, период публикации 2021–2026, доступность полного текста или расширенной аннотации с числовыми результатами.

Поколения методов

Развитие LLM-генерации тестов с 2022 по 2026 год укладывается в три отчётливых поколения архитектурных решений, каждое из которых отвечало на ограничения предыдущего.

Поколение 1 (2022–2023): базовая промпт-генерация. Исходный код тестируемой функции подавался на вход модели, которая генерировала тест в одном обращении. Главные проблемы — высокий процент некомпилируемых тестов и ошибки в ассерциях из-за отсутствия информации о поведении зависимостей. Ранние инструменты на базе Codex (AthenaTest [12]) достигали покрытия фокусного метода на уровне 43,75%.

Поколение 2 (2023–2024): контекстное обогащение и repair-loop. В промпт начали включать зависимости целевого метода, сигнатуры вызываемых функций, документацию. Центральным архитектурным элементом стал цикл исправления: при неудаче теста сообщение об ошибке компиляции или выполнения передавалось обратно в модель для повторной попытки. По данным обзора Чжу и соавт. (2025), 89% исследований использовали prompt engineering как основной метод адаптации LLM [4].

Поколение 3 (2025–2026): агентные системы. Агенты получили доступ к инструментам исполнения кода, анализу покрытия и мутационного тестирования в режиме реального времени; цикл «генерация, выполнение, анализ, регенерация» замкнулся без участия человека. Эта архитектура порождает новые риски, зафиксированные в работах 2025–2026 годов.

Основные фреймворки второго поколения

Три ключевых инструмента второго поколения (2023–2024) демонстрируют, что repair-loop и контекстное обогащение дают измеримый прирост качества. TestPilot (Шефер и соавт., 2023) для JavaScript использует только сигнатуру функции и документацию [9]: на 25 npm-пакетах gpt-3.5-turbo достиг 70,2% медианного покрытия операторов против 51,3% у SBST-инструмента Nessie. ChatUniTest (Чэнь и соавт., 2023/2024) для Java применяет adaptive focal context (динамический отбор релевантных зависимостей в промпт) [3]: покрытие фокусного метода 79,67% против 43,75% у AthenaTest,

превосходство над EvoSuite на 8 из 10 проектов по линейному покрытию. ChatTESTER оптимизирует не покрытие, а корректность ассерций через итеративный рефакторинг с детализированной обратной связью [14]: +34,3% компилируемых тестов и +18,7% тестов с корректными ассерциями по сравнению с базовой ChatGPT.

LLM против EvoSuite: неоднозначность результатов

Данные о сравнительной производительности LLM и EvoSuite существенно зависят от языка программирования, набора проектов и архитектуры инструмента.

Танг и соавт. (2023) сравнили ChatGPT и EvoSuite на Java-проектах по четырём критериям: корректность, читаемость, покрытие кода, обнаружение багов [10]. По покрытию EvoSuite лидирует заметно: 78,91% строк против 40,43% у GPT-4. Однако те же авторы фиксируют качественное преимущество LLM: тесты ChatGPT превосходят EvoSuite по читаемости и практической применимости.

Янг и соавт. (2024) сравнили пять открытых LLM, GPT-4 и EvoSuite на 17 Java-проектах (ASE 2024) [13] и установили, что ни одна модель не доминирует стабильно: EvoSuite выигрывает при максимизации ветвевого покрытия, GPT-4 конкурентоспособен на проектах со сложной логикой. Расхождение с результатами Танга объясняется методологически: разные наборы проектов, языки и стратегии промптирования дают несопоставимые числа, что существенно для интерпретации всей литературы по теме.

Mutation Score: скрытая слабость

Покрытие строк и ветвей представляют необходимые, но недостаточные метрики качества тестов. Mutation score, то есть доля искусственно внесённых ошибок (мутантов), выявленных тестовым набором, считается более строгой метрикой: тест, достигающий строки кода, но не проверяющий возвращаемое значение, не убивает мутанта.

Двараканнат и соавт. (2024) оценили LLM-генерацию на наборе из 7 000 тестов, используя mutation testing как механизм обратной связи [5].

Результаты: 78,5% синтаксической корректности, 67,09% соответствия требованиям, 61,7% покрытия кода, и лишь 18,9% mutation score. Применение дообучения и целенаправленной оптимизации промптов позволяет существенно улучшить mutation score по сравнению с базовой нулевой генерацией [5], однако даже улучшенные инструменты существенно отстают от человеческих тестов по данной метрике.

Обзор Чжу и соавт. (2025) называет «weak fault detection» (слабую способность обнаруживать дефекты) одной из двух нерешённых проблем области наряду с отсутствием стандартизированных бенчмарков [4]. На практике LLM-тесты успешно проходят CI/CD-пайплайн и показывают приемлемое покрытие, одновременно пропуская значительную долю реальных ошибок в коде.

Test Smells

Test smell (структурный или семантический дефект теста, снижающий диагностическую ценность и затрудняющий сопровождение) хорошо изучен применительно к ручному коду. Удредраого и соавт. (2024) провели первое крупномасштабное исследование test smells в LLM-генерируемых тестах, проанализировав 20 505 наборов тестов от GPT-3.5, GPT-4, Mistral 7B и Mixtral 8x7B, а также 14 469 тестов EvoSuite и 779 585 человеческих тестов из 34 635 Java-проектов с использованием инструментов TsDetect и JNose [8].

Исследование выявило два доминирующих типа дефектов.

Assertion Roulette (ассерционная рулетка) — несколько ассерций в одном тест-методе без поясняющих описаний. При падении теста разработчик видит лишь «тест не пройден», но не может определить, какая именно проверка нарушена. Этот smell зафиксирован в более чем 50% тестов при zero-shot промпт-генерации.

Magic Number Test (тест с магическими числами) — жёстко закодированные числовые константы в ассерциях без пояснения их происхождения и смысла. Встречается в 100% тестов на проектах класса высокопроизводительных вычислений (HPC).

Паттерн smells зависит от стратегии промптирования, длины контекста и масштаба модели. Распределение smells в LLM-тестах статистически близко к таковому в человеческом коде; авторы связывают это с утечкой данных из обучающих корпусов.

Нестабильность тестов

Бердт и соавт. провели первое промышленное исследование нестабильности LLM-тестов на четырёх реляционных СУБД (SAP HANA, DuckDB, MySQL, SQLite), используя GPT-4o и Mistral-Large-Instruct-2407 как инструменты амплификации [2]. LLM-тесты демонстрируют лишь незначительно более высокую долю нестабильных тестов по сравнению с ручными, однако 63% нестабильных случаев вызваны одной причиной: тест предполагает фиксированный порядок обхода неупорядоченных коллекций там, где порядок не гарантирован. Здесь проявляется механизм наследования нестабильности: когда repair-loop включает в контекст существующие тесты, модель копирует их проблемные паттерны, воспроизводя нестабильность в новых тестах вместо её устранения.

Избыточное мокирование

При избыточном мокировании тест фактически проверяет заглушку, а не реальное поведение зависимости; интеграционные пути остаются непокрытыми при формально высоком покрытии строк. Гора и Робес (2026) исследовали этот феномен на 1,2 млн коммитов в 2 168 репозиториях (TypeScript, JavaScript, Python) за 2025 год [7]: 36% коммитов от агентов добавляют mock-объекты против 26% от разработчиков-людей. Объяснение структурное: агент не имеет доступа к реальным зависимостям на этапе генерации, и изоляция оказывается самым простым способом получить компилируемый тест, который пройдёт CI.

Концепция и определение

Низкий mutation score [5], test smells [8], flaky tests [2] и over-mocking [7]: четыре независимо зафиксированных дефекта складываются в единую картину, для описания которой авторы вводят концепцию «ложной

безопасности». Под ложной безопасностью понимается ситуация, когда метрики тестирования (прежде всего покрытие строк кода) создают у команды внешне обоснованное, но фактически ложное убеждение в достаточной верифицированности программной системы: не потому, что тесты реально контролируют качество, а потому что они структурно имитируют такой контроль.

Для программной инженерии этот феномен не нов: ложная безопасность фиксировалась и применительно к автоматически сгенерированным EvoSuite-тестам с тривиальными ассерциями. Однако LLM масштабируют проблему: если EvoSuite генерирует нечитаемые тесты, которые воспринимаются разработчиком со скептицизмом, LLM создают читаемые, структурно похожие на ручные тесты, вызывающие доверие.

Механизмы формирования

Все описанные дефекты объединяет одна черта: они создают видимость качества, не обеспечивая его.

Базовая проблема — структурное покрытие без семантической проверки. Тест достигает нужной строки кода, но не проверяет, что именно она возвращает. Over-mocking делает это явным количественно: агент изолирует зависимость вместо того, чтобы её протестировать, и метрика покрытия от этого не страдает. Схожую природу имеет Assertion Roulette: тест с десятком ассерций без пояснений выглядит тщательным, пока не упадёт и разработчик не поймёт, которая из десяти проверок нарушена [8]. Оба случая это не плохие тесты в традиционном смысле, а тесты, которые убедительно имитируют полноценное тестирование.

Происхождение этой имитации объяснимо. LLM воспроизводят канонические структуры тестов из обучающих данных, не рассуждая о том, какие инварианты важны для конкретной системы. Удредраого и соавт. установили, что распределение smells в LLM-тестах статистически близко к таковому в человеческом коде [8], то есть модель воспроизводит привычные паттерны вне зависимости от их пригодности. Контекстная слепота усугубляет

картину: у модели нет доступа к runtime-поведению системы, она не знает, какие входные данные вызывают граничные случаи. Ошибки, проявляющиеся только в специфических условиях, при таком подходе не обнаруживаются систематически.

Бердт и соавт. зафиксировали ещё один механизм, проявляющийся при масштабировании: наследование нестабильности [2]. Когда repair-loop использует существующие тесты как контекст, модель копирует их проблемные паттерны. Нестабильные тесты постепенно отключаются командой, и реальное покрытие расходится с отображаемым без каких-либо предупреждений.

Отечественный контекст

Специализированных отечественных исследований LLM-генерации тестов в доступной литературе авторами не обнаружено. Концептуально близка работа Слекеничса (2025), посвящённая предиктивным ML-моделям приоритизации тестов на промышленных Java/Kotlin-проектах [1]. Несмотря на то что автор исследует не LLM-генерацию как таковую, его результаты концептуально поддерживают аргумент о ложной безопасности: выявленная нелинейная зависимость между плотностью тестов и плотностью дефектов показывает, что оптимальный диапазон покрытия составляет 30–70%, а механический рост покрытия любыми средствами за этими пределами не снижает, а в ряде случаев повышает плотность дефектов. Это эмпирически подтверждает, что метрика покрытия без контроля качества тестов остаётся ненадёжным индикатором надёжности системы.

Консенсус в литературе

В табл. 1 собраны утверждения, подтверждённые несколькими работами.

Утверждение	**Источники**
LLM генерируют более читаемые тесты, чем EvoSuite	[4, 10, 13]

Итеративный repair loop существенно улучшает корректность и компилируемость	[3, 9, 14]
Test smells (Assertion Roulette, Magic Number) преобладают в LLM-тестах	[8, 13]
LLM слабы в обнаружении дефектов (mutation score)	[4, 5]
Дизайн промпта критически влияет на качество генерации	[4, 13]

Таблица 1. Консенсусные утверждения об LLM-генерации тестов

Примечания: (1) избыточное мокирование AI-агентами зафиксировано в единственном исследовании [7] и требует репликации; (2) покрытие кода остаётся предметом разногласий — Танг и соавт. [10] фиксируют преимущество EvoSuite (78,91% vs 40,43% GPT-4), тогда как ChatUniTest [3] превосходит его на 8 из 10 проектов; расхождение методологически обусловлено несопоставимостью бенчмарков.

Пробелы в исследованиях

Чжан и соавт. (2025), проанализировав 115 публикаций, называют отсутствие стандартизированных бенчмарков центральной нерешённой проблемой области [15]. Не менее значимы пробелы в изучении долгосрочной сопровождаемости: все рассмотренные исследования фиксируют качество тестов в момент генерации, но не отслеживают их поведение при рефакторинге и смене зависимостей. За рамками существующих работ остаются российский промышленный контекст, экономическое сопоставление затрат на LLM-генерацию с ручным написанием тестов и инженерные решения для интеграции LLM-генерации в CI/CD без накопления нестабильных тестов.

Заключение

Главное достоинство LLM-генерации тестов (читаемость сгенерированного кода) одновременно является источником её главной опасности. Тест, выглядящий как написанный опытным разработчиком,

вызывает доверие автоматически. Нечитаемые тесты EvoSuite хотя бы не создают иллюзий.

Эмпирические данные 2021–2026 годов фиксируют реальный прогресс: ChatUniTest превосходит EvoSuite по линейному покрытию на 8 из 10 проектов, ChatTESTER даёт +34,3% компилируемых тестов по сравнению с базовой ChatGPT. Вместе с тем mutation score остаётся на уровне 18,9% [5], Assertion Roulette встречается более чем в половине zero-shot тестов [8], агенты систематически over-mock [7]. Высокое покрытие строк и низкий mutation score сосуществуют в одном тесте не как исключение, а как закономерность.

Из этого следует практический вывод. Mutation score и выборочное ревью ассерций должны стать обязательными при массовом использовании LLM-генерации как единственный способ убедиться, что тесты действительно что-то проверяют. Интеграция LLM-инструментов с мутационным тестированием в качестве обратной связи при генерации только начинает разрабатываться, но уже показывает значимые результаты [5].

СПИСОК ЛИТЕРАТУРЫ

1. Слекеничс А.Я. Опыт использования предиктивных ML-моделей при автоматизации тестирования ПО // Universum: Технические науки. 2025. №5. URL: https://cyberleninka.ru/article/n/opyt-ispolzovaniya-prediktivnyh-ml-modeley-pri-avtomatizatsii-testirovaniya-po (дата обращения: 05.06.2026).
2. Berndt A., Bach T., Gemulla R., Kessel M., Baltes S. On the Flakiness of LLM-Generated Tests for Industrial and Open-Source Database Management Systems // 48th International Conference on Software Engineering: Software Engineering in Practice (ICSE SEIP 2026). 2026. URL: https://arxiv.org/abs/2601.08998 (дата обращения: 05.06.2026).
3. Chen Y., Hu Z., Zhi C., Han J., Deng S., Yin J. ChatUniTest: A Framework for LLM-Based Test Generation // Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE

2024). 2024. URL: https://doi.org/10.1145/3663529.3663801 (дата обращения: 05.06.2026).

4. Chu B., Feng Y., Liu K. et al. Large Language Models for Unit Test Generation: Achievements, Challenges, and Opportunities. arXiv:2511.21382, 2025. URL: https://arxiv.org/abs/2511.21382 (дата обращения: 05.06.2026).
5. Dwarakanath A. et al. Effective Test Generation Using Pre-Trained Large Language Models and Mutation Testing // Information and Software Technology. 2024. Vol. 171. Art. 107468. URL: https://doi.org/10.1016/j.infsof.2024.107468 (дата обращения: 05.06.2026).
6. Fraser G., Arcuri A. EvoSuite: Automatic Test Suite Generation for Object-Oriented Software // Proceedings of the 19th ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE 2011). New York: ACM, 2011. P. 416–419. URL: https://doi.org/10.1145/2025113.2025179 (дата обращения: 05.06.2026).
7. Hora A., Robbes R. Are Coding Agents Generating Over-Mocked Tests? An Empirical Study // Proceedings of the 23rd International Conference on Mining Software Repositories (MSR 2026). 2026. URL: https://arxiv.org/abs/2602.00409 (дата обращения: 05.06.2026).
8. Ouédraogo W.C., Li Y., Dang X. et al. Test Smells in LLM-Generated Unit Tests. arXiv:2410.10628, 2024. URL: https://arxiv.org/abs/2410.10628 (дата обращения: 05.06.2026).
9. Schäfer M., Nadi S., Eghbali A., Tip F. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation. arXiv:2302.06527, 2023. URL: https://arxiv.org/abs/2302.06527 (дата обращения: 05.06.2026).
10. Tang Y., Liu Z., Zhou Z., Luo X. ChatGPT vs SBST: A Comparative Assessment of Unit Test Suite Generation. arXiv:2307.00588, 2023. URL: https://arxiv.org/abs/2307.00588 (дата обращения: 05.06.2026).

11. Tassey G. The Economic Impacts of Inadequate Infrastructure for Software Testing. Gaithersburg: National Institute of Standards and Technology, 2002. 309 p. URL: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf (дата обращения: 05.06.2026).
12. Tufano M., Drain D., Svyatkovskiy A., Deng S., Sundaresan N. Unit Test Case Generation with Transformers and Focal Context. arXiv:2009.05617, 2020. URL: https://arxiv.org/abs/2009.05617 (дата обращения: 05.06.2026).
13. Yang L., Yang C., Gao S. et al. On the Evaluation of Large Language Models in Unit Test Generation // 39th IEEE/ACM International Conference on Automated Software Engineering (ASE 2024). 2024. URL: https://arxiv.org/abs/2406.18181 (дата обращения: 05.06.2026).
14. Yuan Z., Lou Y., Liu M. et al. No More Manual Tests? Evaluating and Improving ChatGPT for Unit Test Generation. arXiv:2305.04207, 2023. URL: https://arxiv.org/abs/2305.04207 (дата обращения: 05.06.2026).
15. Zhang Q., Fang C., Gu S. et al. Large Language Models for Unit Testing: A Systematic Literature Review. arXiv:2506.15227, 2025. URL: https://arxiv.org/abs/2506.15227 (дата обращения: 05.06.2026).