УДК 004.052
Оригинальная статья
https://doi.org/10.21683/1729-2646-2025-25-4-61-68

Выявление системных неисправностей в программно-аппаратных комплексах на основе интеллектуальных технологий

Detecting system faults in hardware and software systems using intelligent solutions

Панков И.А.[1]*, Аверченко А.П.[1], Панков Д.А.[2]*
Pankov I.A.[1]*, Averchenko A.P.[1], Pankov D.A.[2]*

[1] *Омский государственный технический университет, Омск, Российская Федерация*
[2] *ООО «ЛАНИТ-ТЕРКОМ», Санкт-Петербург, Российская Федерация*
[1] *Omsk State Technical University, Omsk, Russian Federation*
[2] *LANIT-TERKOM LLC, Saint Petersburg, Russian Federation*
* *E-mail: pankovDDD@yandex.ru, pankov99ai@yandex.ru*

Панков И.А.

Аверченко А.П.

Панков Д.А.

Резюме. *Представлена система выявления неисправностей в распределенных программно-аппаратных комплексах, основанная на наборе интеллектуальных технологий. Подход объединяет динамическое тестирование (фаззинг), корректируемое большими языковыми моделями, а также анализ шаблонов уязвимостей известных баз знаний MITRE и OWASP для выявления программных ошибок, способствующих проведению потенциальных атак. Предложенная архитектура оперативно диагностирует отказы и сбои, локализует их причину и автоматически эскалирует инцидент системному администратору. Практическая значимость решения подтверждена экспериментально по таким параметрам, как среднее время обнаружения ошибок, охват кода, количество обнаруженных дефектов.*

Abstract. *The paper presents a system for detecting faults in distributed software and hardware systems that is based on a set of intelligent technologies. The method combines dynamic testing (fuzzing) enhanced with large language models, as well as analysis of vulnerability patterns of the well-known MITRE and OWASP knowledge bases to identify software errors that enable potential attacks. The proposed architecture promptly diagnoses failures and faults, localizes their causes and automatically escalates the incident to the system administrator. The practical significance of the solution is confirmed experimentally in terms of such parameters as the average error detection time, code coverage, and the number of detected defects.*

Ключевые слова: *обнаружение неисправностей, распределенные комплексы, отказы и сбои, фаззинг, большие языковые модели.*
Keywords: *fault detection, distributed systems, failures and faults, fuzzing, large language models.*

Для цитирования: *Панков И.А., Аверченко А.П., Панков Д.А. Выявление системных неисправностей в программно-аппаратных комплексах на основе интеллектуальных технологий // Надежность. 2025. №4. С. 61-68. https://doi.org/10.21683/1729-2646-2025-25-4-61-68*
For citation: *Pankov I.A, Averchenko A.P., Pankov D.A. Detecting system faults in hardware and software systems using intelligent solutions. Dependability 2025;4: 61-68. https://doi.org/10.21683/1729-2646-2025-25-4-61-68*

Поступила: *27.08.2025* / **После доработки:** *11.09.2025* / **К печати:** *28.09.2025*
Received on: *27.08.2025* / **Revised on:** *11.09.2025* / **For printing:** *28.09.2025*

Введение

Современные распределенные программно-аппаратные комплексы управления и связи представляют собой сложные системы, функционирование которых зависит от различных внутренних и внешних факторов. Система содержит несколько интерфейсов связи, таких как локальные сети, радиосвязь, беспроводные протоколы; также она состоит из центра управления и нескольких связных узлов (подсистем), которые географически распределены (рис. 1). Контроль устойчивости программно-аппаратного комплекса (ПАК) к отказам и сбоям в реальном масштабе времени является важной задачей для выполнения требований, заложенных в техническое задание на проектирование. Цель статьи – повысить эффективность поиска неисправностей и потенциальных уязвимостей в программах для ПАК за счет внедрения комплекса интеллектуальных технологий. В распределенном ПАК есть специальная программа, которая имитирует сбои и отказы, которая делает это не для одного устройства, а сразу для подсистем. В процессе разработки и эксплуатации ПАК для проверки его устойчивости применяют три основных метода: имитация неисправностей на программном уровне, сканирование аппаратного состояния

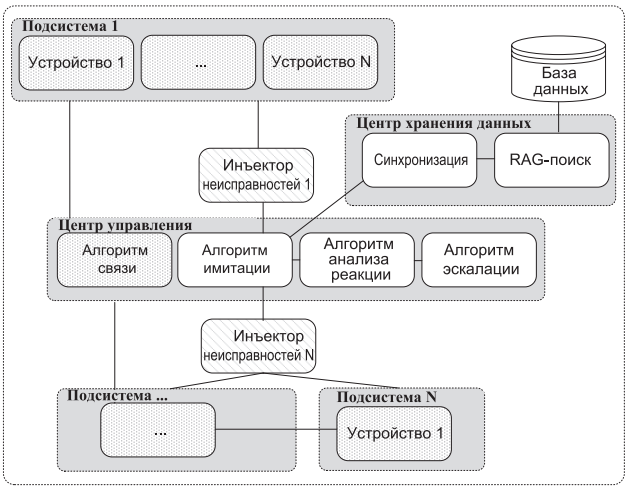


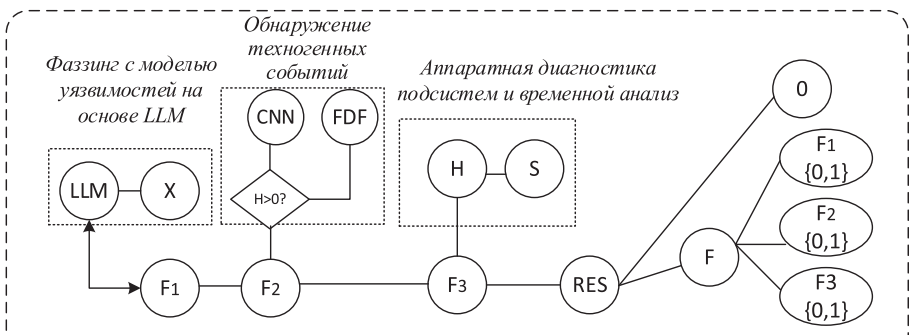
Рис. 1. Структурная схема ПАК

устройств, акустический мониторинг (как дополнительный способ контроля техногенных событий). Эти методы позволяют собирать и анализировать комплексную информацию обо всех устройствах в подсистеме. Инъекторы неисправностей объединены в единую локальную сеть, что дает возможность централизованно управлять тестированием и передавать данные в модуль анализа и детектор атак. Традиционные методы диагностики часто недостаточно эффективны для оперативного определения причин отказов.

Проверка устойчивости к отказам и сбоям необходима ПАК не только в процессе разработки, но и во время эксплуатации. Это связано с двумя основными проблемами: структура распределенной системы постоянно меняется, когда добавляются или исключаются устройства, что требует непрерывного мониторинга и диагностики. Также стандартных средств диагностики в ряде случаев недостаточно, чтобы четко разделить программные и аппаратные отказы из-за высокой взаимосвязи компонентов. В данной статье наиболее подробно рассмотрен вопрос эффективного поиска неисправностей, которые вызваны программными дефектами. Для поиска программных ошибок комплекса используется фаззинг, которому для получения наилучших результатов охвата алгоритмов работы устройств необходимы входные данные. Часто получение таких данных затруднено в связи с особенностью тестирования или сложной структурой протоколов обмена [2] (рис. 2). Для генерации входных данных эффективно использовать алгоритмы больших языковых моделей (Large Language Models, LLM).

Предложенная система имитации позволяет детализировать причины отказов за счет анализа подсистем ПАК, которые возникают в процессе работы оборудования. Контроль осуществляется алгоритмом анализа реакции на имитируемые неисправности с помощью фаззинга. Алгоритм обрабатывает и интерпретирует данные на основе работы нейросети, а также методов нечеткого логического вывода, что обеспечивает более точное локализации дефекта (рис. 3).

Поскольку генерация данных применяет алгоритмы LLM, то следует уточнить, что подготовка структуры



F1 – программные ошибки, F2 – техногенные события, F3 – аппаратные отказы и сбои, H – признак аппаратной ошибки LLM – Алгоритм внесения искажений на основе LLM модели. FDF–нечеткий логический вывод, CNN–сверточная нейросеть. RES – множество {F1, F2, F3}, где ошибки отсутствует, либо одна или несколько видов неисправностей

Рис. 2. Схема фаззинга ПАК

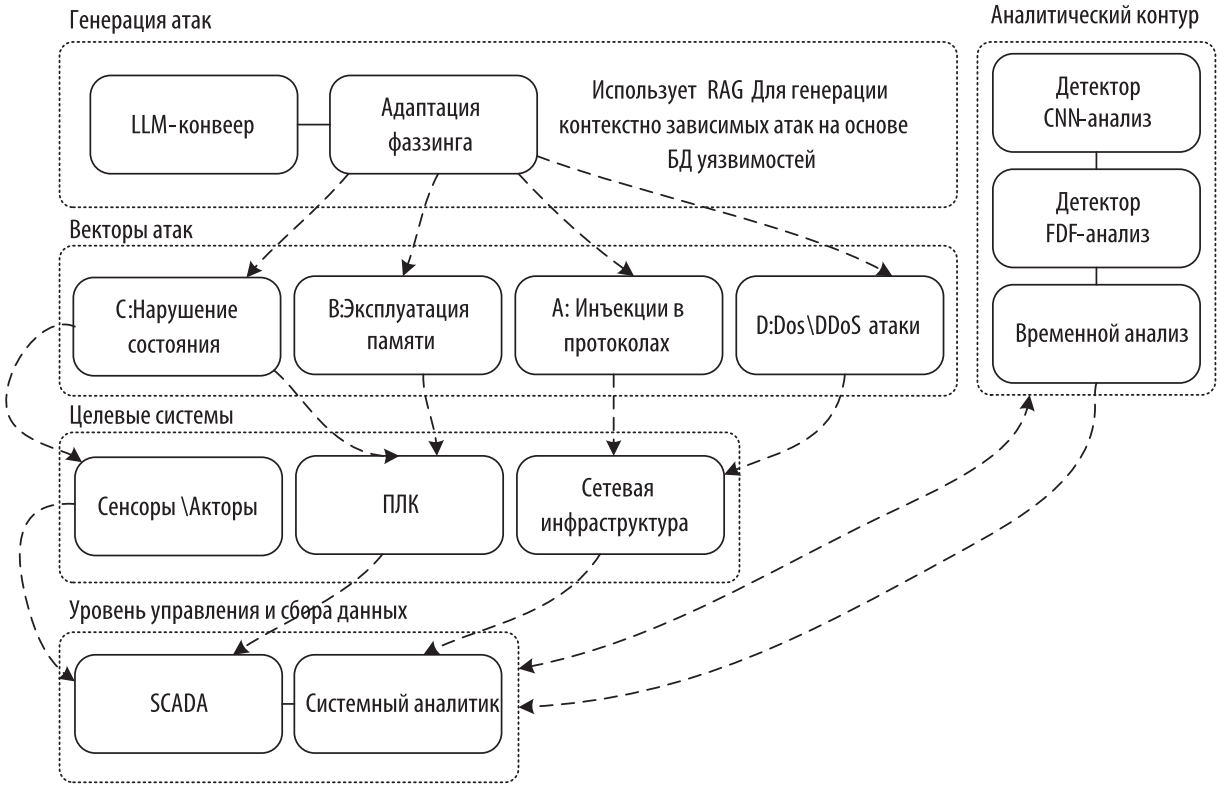


Рис. 3. Схема формирования векторов атак для подсистем ПАК с использованием базы знаний CVE

данных на вход алгоритмов фаззинга происходит с помощью алгоритма генерации, дополненного поиском – Retrieval Augmented Generation или RAG [2].

RAG-подход является формирование так называемого «вектора атаки», представляющего собой конкретное тестовое воздействие, нацеленное на проверку устойчивости системы к угрозам из баз знаний уязвимостей информационной безопасности CVE. Для этого RAG выбирает известные уязвимости из базы знаний для используемых в ПАК программных средствах и аппаратных модулях. С помощью семантического поиска, основанного на преобразовании текстовых запросов в векторные представления, осуществляется сопоставление с фрагментами информации из базы знаний по принципу семантической близости. В результате извлекаются наиболее релевантные описания уязвимостей, эксплойтов (программных модулей для эксплуатации уязвимостей) и атакующих шаблонов.

Применение заданной модели эффективно как для имитации неисправностей в памяти устройств, так и для искажения данных протоколов обмена между устройствами. Такой контекст является специфичным для текущей цели тестирования и существенно расширяет возможности LLM по генерации точных и применимых в конкретной ситуации тестовых данных. Генерация перестает быть абстрактной: модель синтезирует конкретные команды, сетевые пакеты, входные данные для фаззинга (направленного случайного тестирования) или сценарии эксплуатации, ориентированные на выявленные уязвимости.

1. Архитектура конвейера программного поиска дефектов

Поиск дефектов программ в ПАК реализован в виде конвейера, включающего обработку и генерацию данных, проведение фаззинга и анализ результатов с контролем выполнения по метрикам объема тестирования. Фаззинг представляет собой метод динамического тестирования программного обеспечения, при котором в систему подаются случайно сгенерированные или модифицированные входные данные с целью вызвать сбои, переполнения буфера, ошибки обработки или другое аномальное поведение программы [3]. В контексте распределенных систем этот подход позволяет проверять как отдельные компоненты, так и взаимодействие между ними, включая сетевые протоколы, интерфейсы, форматы обмена данными и аппаратные драйверы [3, 4]. Общий алгоритм включает подготовку данных с определением режимов испытания устройства согласно шаблону спецификации (*шаблон 1*), внесение отказов и сбоев в конечное устройство с изученным программным кодом с помощью техники фаззинга (*шаблон 2*). После чего осуществляется определение тестовых испытаний по имитации неисправностей на основе нечеткого логического вывода. Тестирование устройства производится путем искусственного внесения ошибок в программное и аппаратное обеспечение проверяемых устройств на основе тестовой документации и статистики ошибок, которые были обнаружены в период тестирования.

Для программно-аппаратных систем существует ряд ограничений по приросту объема тестирования в

процентах протестированного кода, который сложно обойти без применения дополнительных инструментов. Фаззинг используется совместно с алгоритмом LLM для улучшения процесса имитации неисправностей, что позволяет увеличить скорость и объем тестируемого кода. Алгоритм LLM формирует данные на вход работы алгоритма фаззинга, а также выходные данные фаззинга, которые поступают на вход работы алгоритма LLM для генерации новых шаблонов, которые позволяют проверить как устойчивость к обычных ошибкам программного обеспечения, так и для проверки устойчивости к модификации известных шаблонов атак [5, 6]. Приведем формальное описание процесса поиска ошибок программ ниже.

$\sum_x b_x \to \max$, где b_x – кумулятивный набор дефектов за счет применения тестовых наборов в распределенной программно-аппаратной системе для набора атакующих запросов на основе уязвимостей и дефектов устройств.

Для $c_{i,j}$ – i-й сбой или отказ при выполнении j-го искажения данных;

$A_{m(t)} = \{a_{q1}, a_{q2}, ..., a_{|a_{q|O|}}\}$ – набора атакующих запросов на основе уязвимостей и дефектов устройств;

$R_{m(t)} = \{r_{q1}, r_{q2}, ..., r_{|a_{q|O|}}\}$ – наборов реализованных дефектов с помощью объединения оригинальных шаблонов (O) и модифицированных;

$P = \{o_1, o_2, ..., o_{|O|}\} \cup \{g_1, ..., g_n\}$ – множество тестов программы для устройства ПАК.

Особенность подхода заключается также в способности к генерации сложных, многоэтапных сценариев атак, имитирующих реальные тактики, техники и процедуры, применяемые злоумышленниками. Такие сценарии трудно воспроизвести с помощью классических методов случайного фаззинга. Важным преимуществом является и высокая адаптивность RAG: по мере получения обратной связи от системы, например, через сигналы от сверточных нейронных сетей (Convolutional Neural Networks, CNN), временной анализ или методы нечеткой логики, происходит динамическое обновление контекста и формулировка новых запросов. Это позволяет уточнять и углублять исследование потенциальных векторов атак.

Процесс тестирования с использованием RAG включает несколько последовательно выполняемых этапов. Сначала, на основе полученного контекста, LLM генерирует конкретные тестовые воздействия [7, 8]. Эти воздействия затем автоматически применяются к тестируемой системе. Цикл тестирования обеспечивает улучшение качества воздействия и повышение вероятности выявления уязвимостей.

Для проведения исследования с помощью техники фаззинга представим в виде конвейера обработки для создаваемого ПАК (рис. 4).

Интеграция RAG осуществляется на этапе активного тестирования системы, где он взаимодействует с адаптивным фаззингом, управляемым LLM, а также инструментами временного анализа. Таким образом, обеспечивается интеллектуальное, контекстно-зависимое наполнение процесса генерации тестовых воздействий. Это напрямую способствует повышению точности классификации инцидентов информационной безопасности, снижению времени обнаружения целевых атак и увеличению охвата потенциальных уязвимостей.

Следует подчеркнуть, что целью использования RAG в данной архитектуре является не извлечение готовых ответов, а именно генерация интеллектуальных тестовых воздействий, синтезируемых на основе специализированного контекста, извлеченного из базы знаний. Вектор атаки представляет собой конкретную команду, пакет данных или скрипт, сгенерированный языковой моделью на основе актуальных сведений об уязвимостях. Автоматизированный цикл RAG совместно с LLM обеспечивает высокий уровень покрытия атакующих сценариев для исследуемых систем.

2. Применение алгоритмов LLM для улучшающих объема найденных ошибок с помощью фаззинга

Работа LLM в сочетании с фаззингом строится следующим образом. Сначала проводится сбор и предварительная обработка данных – собираются образцы входных сообщений, логи работы системы, специфи-

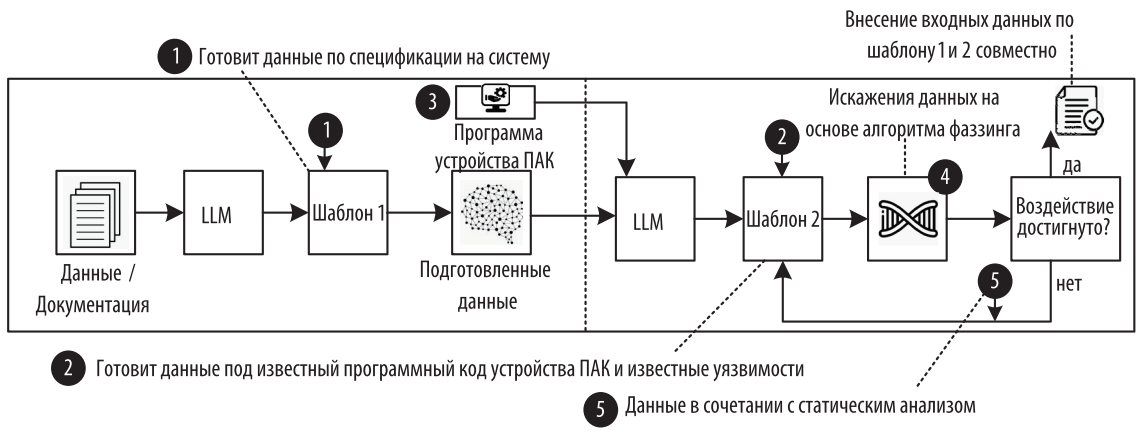


Рис. 4. Структурная схема конвейера генерации данных для устройств ПАК с использованием алгоритма LLM

кации протоколов и API. Эти данные используются для дообучения или тонкой настройки языковой модели, чтобы адаптировать ее под конкретную предметную область [9, 10]. Затем модель интегрируется в фаззинг-систему, которая вместо случайной генерации использует выход работы алгоритма LLM для создания более реалистичных и содержательных тестовых данных. Ключевое нововведение – LLM L определяет условную вероятность $P_L(i \mid C)$ для генерации нового входа i при наличии контекста C. Новый вход (i_{new}): Данные, которые будут поданы в программу P. Контекст C**:** информация, доступная модели для принятия решения. Контекст может включать:

• Исходный код программы P (полностью или частично).

• Начальный набор валидных входов (corpus S).

• Историю предыдущих сгенерированных входов $\{i_1, i_2, ..., i_{t-1}\}$.

• Информацию обратной связи: покрытие кода, отчеты о сбоях.

Таким образом, на каждом шаге t фаззинга LLM генерирует новый вход: $i_t \sim P_L(i \mid C_t)$

$$\max_{\theta} \mathrm{E}\left[\sum_{t=1}^{T} O(i_t)\right],$$

θ – это параметры LLM (или параметры запроса), для которого проводится оптимизация;

i_t – вход, сгенерированный на шаге t согласно $P_L(i \mid C_t; \theta)$;

$O(i_t)$ – результат проверки входа решающим алгоритмом;

E – математическое ожидание, которое учитывает вероятностный характер генерации входов.

Модель обучается на примерах корректных сообщений, передаваемых между компонентами системы, после чего генерирует тестовые данные, которые не только соответствуют синтаксису протокола, но и учитывают его семантические особенности. На этапе выполнения фаззинга система запускает тестовые случаи, сгенерированные моделью, после чего отслеживает реакцию программно-аппаратного комплекса. При этом используется обратная связь от мониторинговых и аналитических модулей, которые фиксируют покрытие кода, возникшие ошибки и нештатное поведение системы (рис. 5).

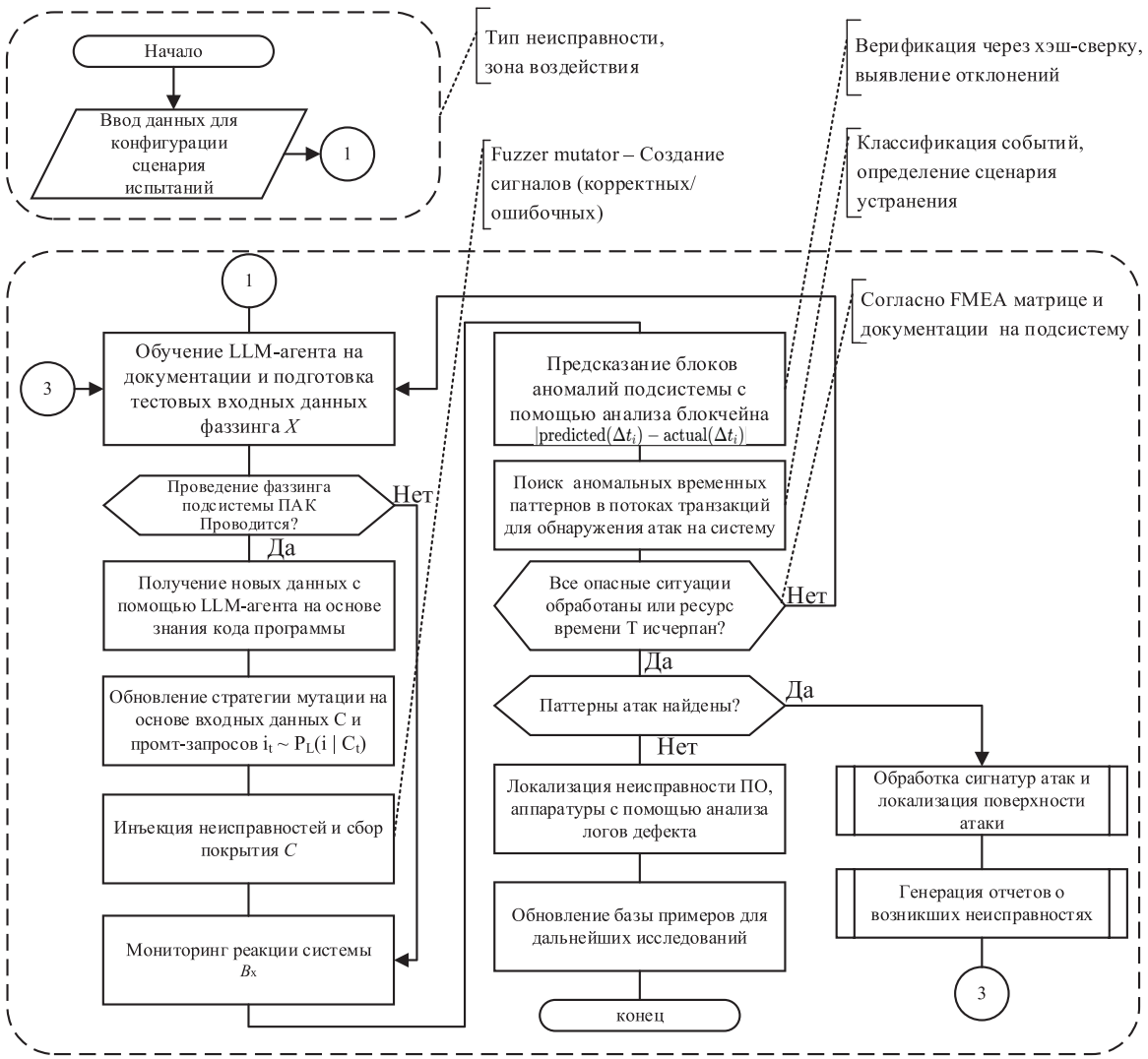


Рис. 5. Алгоритм имитации неисправностей на основе фаззинга с LLM коррекцией вектора атаки

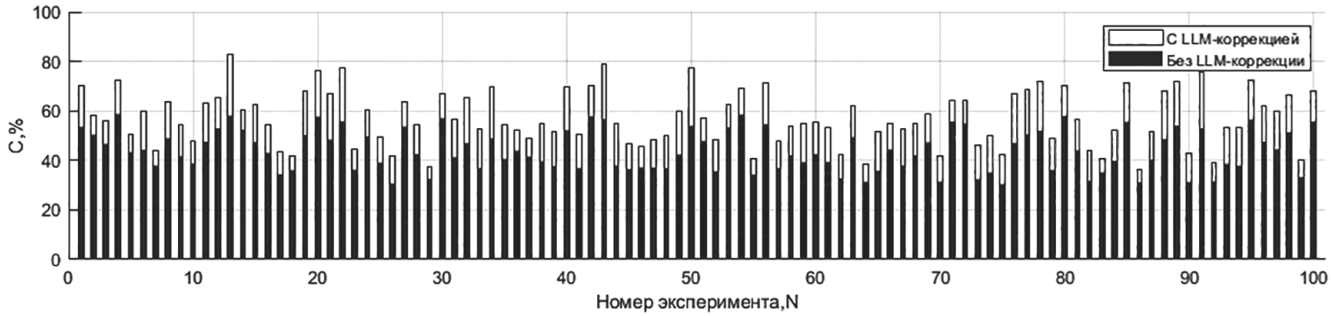


Рис. 6. Сравнение результатов покрытия кода *C*, %, алгоритмом фаззинга с LLM-коррекцией с алгоритмом фаззинга без LLM-коррекции.

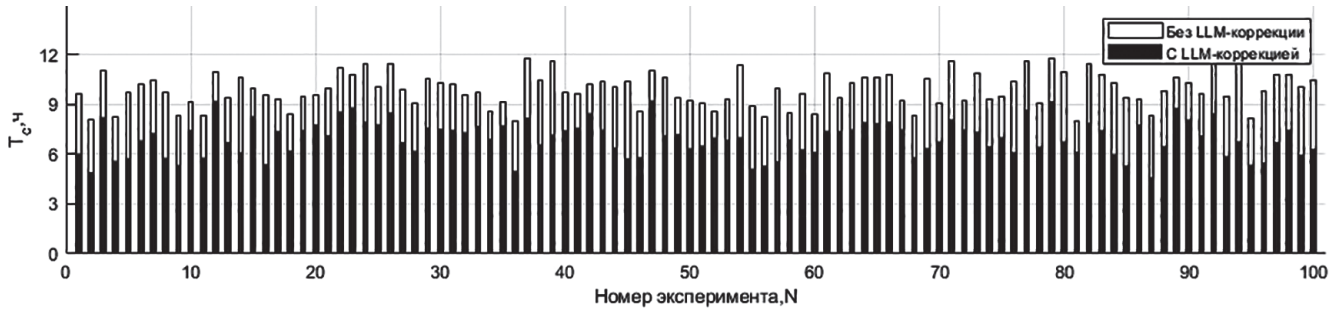


Рис. 7. Сравнение времени выполнения T_C, %, алгоритмом фаззинга с LLM-коррекцией с алгоритмом фаззинга без LLM-коррекции.

Эта информация поступает обратно к LLM, которая корректирует свои дальнейшие генерации, стремясь достигнуть максимального атакующего воздействия на исследуемую систему в виде количества отказов, сбоев и выявленных ошибок при минимальном количестве тестовых запусков. Кроме того, LLM используется для анализа результатов фаззинга. Модель способна интерпретировать логи, классифицировать типы найденных ошибок, выделять повторяющиеся паттерны и предлагать возможные пути исправления уязвимостей. Это особенно ценно в распределенных системах, где количество потенциальных точек отказа велико, а диагностика проблем требует глубокого понимания контекста.

Еще одной важной стороной использования LLM в фаззинге является возможность моделирования атаки на систему [11, 12]. Языковая модель может генерировать не просто одиночные атаки, а такие последовательности, которые имитируют цепочки вредоносных сценариев – например, попытки внедрения кода, перехвата управления, изменения состояния системы с захватом управления.

Такой подход усиливает фаззинг за счет привлечения знаний о современных методах эксплуатации уязвимостей и позволяет заранее находить и устранять потенциально опасные функции в системе [13].

Таким образом, интеграция языковых моделей в процесс фаззинга значительно повышает его практическую значимость, позволяя проводить тестирование более направлено и адаптивно (рис. 6 и 7) для тестирования подсистемы ПАК с тремя устройствами в условиях реальной эксплуатации. Сочетание RAG для интеллектуального доступа к данным, базы знаний MITRE для анализа угроз и базы знаний OWASP для обеспечения безопасности приложений, позволяет создать комплексную систему управления ПАК, которая не только автоматизирует рутинные задачи, но и обеспечивает понимание структуры данных и поведения системы в условиях постоянно меняющихся киберугроз [14, 15].

Заключение

Предложенная в работе система выявления неисправностей для распределенных ПАК основана на использовании фаззинга совместно с подготовкой данных тестирования с применением алгоритмов больших языковых моделей, что позволило повысить устойчивость распределенного ПАК к отказам и сбоям за счет увеличения количества обнаруженных дефектов за заданное время тестирования и последующее устранение найденных программных дефектов. Формализованный процесс фаззинга, валидированный с помощью метрик времени тестирования и объема проверенного кода соответствует международным и отечественным стандартам качества и надежности для электронных устройств. Сравнительный анализ с классическими методами имитации неисправностей показал, что фаззинг совместно с интеллектуальными технологиями, обеспечивает проверку на наличие уязвимостей из баз знаний MITRE и OWASP, что позволяет сокращает временные и ресурсные затраты на устранение дефектов и время работы служб информационной безопасности.

Возможность сигнализации об полученных ошибках администратору системы обеспечивает обратную связь для разработчиков ПАК в процессе эксплуатации. Экспериментальные исследования подтвердили практическую значимость разработанного решения. Было достигнуто увеличение скорости обнаружения ошибок до двух раз и увеличение покрытия кода до 20%, что способствует снижению числа уязвимостей, включая «уязвимости нулевого дня».

Использование представленных интеллектуальных технологий совместно с другими видами анализа, таких как аппаратная диагностика и акустический анализ, будет рассмотрено подробнее в следующих работах и позволит выполнять выявление дефектов более эффективно за счет всестороннего анализа подсистем ПАК.

Список литературы

1. Meng Ruijie, Mirchev Martin, Böhme Marcel et al. Large Language Model guided Protocol Fuzzing. 2024. DOI: 10.14722/ndss.2024.24556

2. Elnaggar R., Delgado B., Fung J. M. RAG-Based Fuzzing of Cross-Architecture Compilers. URL: https://arxiv.org/abs/2504.08967 (дата обращения: 13.07.2025).

3. Панков И.А., Панков Д.А., Денисова Л.А. Автоматизация разработки и тестирования цифровых систем связи с многоуровневой архитектурой // Автоматизация в промышленности. 2023. № 1. С. 31–35.

4. Панков И.А., Панков А.П., Панков Д.А. и др. Перспективы использования FMEA-анализов для высокоответственных технических систем // Известия Тульского государственного университета. Технические науки. 2024. № 3. С. 26–31.

5. Панков И.А. Выявление дефектов при тестировании алгоритмов цифровых устройств на базе ПЛИС // Известия Тульского государственного университета. Технические науки. 2023. № 11. С. 277–280.

6. Панков И.А., Панков Д.А. Обнаружение системных дефектов цифровых устройств при имитации неисправностей с применением // Надежность. 2023. Т. 23. № 4. С. 51–58.

7. Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian et al. Fuzz4All: Universal Fuzzing with Large Language Models. 2024. URL: https://arxiv.org/abs/2308.04748 (дата обращения: 29.09.2025).

8. Zafar A., Wajid B., Akram B. A hybrid fault diagnosis architecture for Wireless Sensor Networks // ICOSST. 2015. DOI: 10.1109/ICOSST.2015.7396395

9. Heracleous C., Keliris C., Panayiotou C. et al. Fault diagnosis for a class of nonlinear uncertain hybrid systems // Nonlinear Analysis: Hybrid Systems. 2022. Vol. 44. P. 101137. DOI: 10.1016/j.nahs.2021.101137

10. Pan Z., Fu Y., Guo H. et al. Analysis of Covert Attacks Using False Data against Network Control Systems: Three Case Studies // Syst Sci Complex. 2023. Vol. 36. Pp. 1407–1422.

11. Reber A., Cha S.K. Optimizing Fuzzing Seed Selection // Proceedings of the 23rd USENIX Security Symposium. San Diego, CA, 2014. Pp. 861–875.

12. Fibich C., Tauner S., Rössler P. et al. FIJI: Fault InJection Instrumenter // EURASIP Journal on Embedded Systems. 2019. DOI: 10.1186/s13639-019-0088-7

13. Trippel T., Shin K.G. Fuzzing Hardware as Software // USENIX: [сайт]. 2022. URL: https://www.usenix.org/system/files/sec22-trippel.pdf (дата обращения: 13.07.2025).

14. Панков Д.А., Денисова Л.А. Контроль и диагностика отказов программно-аппаратных комплексов // Омский научный вестник. 2018. № 2. С. 128–130.

15. Панков Д.А., Денисова Л.А. Проектирование аппаратно-программного комплекса: определение объема тестовых испытаний микропроцессорных устройств // Автоматизация в промышленности. 2020. № 12. С. 23–29.

References

1. Meng R., Mirchev M., Böhme M. et al. Large Language Model guided Protocol Fuzzing. Network and Distributed System Security (NDSS) Symposium 2024; San Diego (CA, USA). DOI: 10.14722/ndss.2024.24556

2. Elnaggar R., Delgado B., Fung J. M. RAG-Based Fuzzing of Cross-Architecture Compilers. (accessed 13.07.2025). Available at: https://arxiv.org/abs/2504.08967.

3. Pankov I.A., Pankov D.A., Denisova L.A. [Automation of development and testing of digital communication systems with multilevel architecture]. *Avtomatizatsiya v promyshlennosti* 2023;1:31-35. (in Russ.)

4. Pankov I.A., Pankov A.P., Pankov D.A. et al. Prospects for the use of FMEA analyzes for highly critical technical systems. *Izvestiya Tula State University* 2024;3:26-31. (in Russ.)

5. Pankov I.A. Detecting defects when testing algorithms digital devices based on FPGA. *Izvestiya Tula State University* 2023;11:277-280. (in Russ.)

6. Pankov D.A., Pankov I.A. Detecting system defects of digital devices in the course of malfunction imitation using fuzzing. *Dependability* 2023;23(4):51-58. (in Russ.)

7. Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian et al. Fuzz4All: Universal Fuzzing with Large Language Models. 2024. (accessed 29.09.2025). Available at: https://arxiv.org/abs/2308.04748.

8. Zafar A., Wajid B., Akram B. A hybrid fault diagnosis architecture for Wireless Sensor Networks. ICOSST; 2015. DOI: 10.1109/ICOSST.2015.7396395

9. Heracleous C., Keliris C., Panayiotou C. et al. Fault diagnosis for a class of nonlinear uncertain hybrid systems. *Nonlinear Analysis: Hybrid Systems* 2022;44:101137. DOI: 10.1016/j.nahs.2021.101137

10. Pan Z., Fu Y., Guo H. et al. Analysis of Covert Attacks Using False Data against Network Control Systems: Three Case Studies. *Syst Sci Complex* 2023;36:1407-1422.

11. Reber A., Cha S.K. Optimizing Fuzzing Seed Selection. Proceedings of the 23rd USENIX Security Symposium. San Diego (CA, USA); 2014. Pp. 861-875.

12. Fibich C., Tauner S., Rössler P. et al. FIJI: Fault InJection Instrumenter. EURASIP Journal on Embedded Systems 2019. DOI: 10.1186/s13639-019-0088-7

13. Trippel T., Shin K.G. Fuzzing Hardware as Software. (accessed 13.07.2025). Available at: https://www.usenix.org/system/files/sec22-trippel.pdf

14. Pankov D.A., Denisova L.A. Control and diagnostics of faults in hardware-software complex. *Omsk Scientific Bulletin* 2018;2(158):128-133. (in Russ.) DOI:https://doi.org/10.25206/1813-8225-2018-158-128-133.

15. Pankov D.A., Denisova L.A. [Designing a software and Hardware system: defining the scope of testing of computer-based devices]. *Avtomatrizatsiya v promyshlennosti* 2020;12:23-29. (in Russ.)

Сведения об авторах

Панков Илья Анатольевич – аспирант ОмГТУ, pankov99ai@yandex.ru.

Аверченко Артем Павлович – аспирант ОмГТУ, руководитель СКБ «Цифровая обработка сигналов на ПЛИС» ОмГТУ.

Панков Денис Анатольевич – руководитель проектов и системный аналитик ООО «ЛАНИТ-ТЕРКОМ», участник программного комитета по стандартизации информационных технологий, канд. техн. наук. pankovDDD@yandex.ru.

About the authors

Ilia A. Pankov, postgraduate student, OmSTU, pankov99ai@yandex.ru.

Artem P. Averchenko, postgraduate student, OmSTU, head of the Digital Signal Processing using FPGA SDB, OmSTU.

Denis A. Pankov, Project Manager and System Analyst, LANIT-TERKOM LLC, Member of the Program Committee for Information Technology Standardisation, Candidate of Engineering, pankovDDD@yandex.ru.

Вклад авторов

Панков Илья Анатольевич провел экспериментальные исследования с применением фаззинга и нечеткой логики для выявления ошибок ПО и техногенных событий.

Аверченко Артем Павлович предложил идеи по имитации неисправностей для программно-аппаратных модулей на ПЛИС.

Панков Денис Анатольевич разработал архитектуру системы имитации неисправностей и предложил комбинацию техники фаззинга совместно с большими языковыми моделями для улучшения объема найденных ошибок ПО и уязвимостей.

Конфликт интересов

Авторы заявляют об отсутствии конфликта интересов.