The American Journal of Applied Sciences
ISSN 2689-0992 | Open Access

TYPE Original Research
PAGE NO. 96-103
DOI 10.37547/tajas/Volume07Issue10-11

Methods for Analysis and Classification of Errors in Automated Tests Using Modern LLM Models

OPEN ACCESS

SUBMITTED 28 August 2025
ACCEPTED 18 September 2025
PUBLISHED 19 October 2025
VOLUME Vol.07 Issue 10 2025

CITATION
Taras Buriak. (2025). Methods for Analysis and Classification of Errors in Automated Tests Using Modern LLM Models. The American Journal of Applied Sciences, 7(10), 96–103. https://doi.org/10.37547/tajas/Volume07Issue10-11

Taras Buriak
Software Development Engineer in Test, USA

Abstract: This article presents an overview of methods for analyzing and classifying errors in automated tests using modern language models. The research is based on a systematization of international publications that examine the solutions RCACopilot, LogLLM, FlakyDoctor, and LogGPT. It is shown that these approaches differ in their architectural solutions and task formulations: classification of incident root causes, anomaly detection in logs, repair of flaky tests, and real-time log interpretation. The study identifies specific data preparation and training strategies that determine the models' effectiveness. The presented metrics demonstrate high accuracy and practical applicability but also point to significant limitations. Among them are a dependence on monitoring infrastructure and computational resources, sensitivity to prompt parameters, and weak results in repairing NOD-type tests. The analysis showed that integrating the models into existing pipelines with filtering and validation allows for minimizing risks and increasing the reliability of the solutions. Practical implementation experience is noted, which confirmed an increase in the stability of test runs and a reduction in regression time. The article will be useful for researchers and practitioners in the fields of software engineering, automated testing, and quality assurance.

Keywords: language models, automated testing, log analysis, anomaly detection, test repair, software quality.

Introduction

Modern software systems include thousands of automated tests that accompany the entire

development and operation cycle. The number of tests is continuously growing, and with it, the volume of errors recorded in logs and reports is also increasing. It is becoming impossible for development and quality engineering teams to conduct a full manual analysis of this data. The errors are complex in nature. Some are related to infrastructure, some to incorrect code logic, and some to the execution environment. Under these conditions, traditional methods for finding the causes of errors are no longer sufficient. They require a lot of time and yield limited results. The emergence of large language models has opened up new possibilities for analysis. These models can interpret textual information, identify patterns, and draw conclusions based on large datasets. Their application in the field of software system testing is a logical next step.

Despite the potential, the use of modern language models in the field of error analysis faces a number of difficulties. The data from automated tests is diverse in form and content. It includes both structured elements and textual descriptions that may contain ambiguous wording [4]. This complicates processing and requires additional preparation stages. Errors in tests are classified by a multitude of characteristics, from anomalies in logs to flaky tests with inconsistent execution results. Automatically distinguishing such cases remains a difficult task [11]. Modern language models require significant computational resources. Their integration into industrial testing processes is associated with the high cost of training and maintenance. Finally, the question of trust in the results remains. Although language models show high accuracy, their decisions are not sufficiently transparent and require additional interpretation. Developers and engineers need explainability to understand the causes of the errors found and to apply corrective measures [1].

The objective of this study is to analyze existing approaches to the application of large language models for the analysis and classification of errors in automated tests, to identify the architectural, logical, and methodological aspects of their use in software quality assurance systems, and to outline the prospects for the transformation of testing processes in the context of the growing complexity of software systems and increasing data volumes.

Materials And Methods

This study is based on the methodology of a systematic analytical review of modern approaches to using large language models for the analysis and classification of errors in automated tests. The primary method is the thematic synthesis of architectures, algorithms, and applied solutions presented in peer-reviewed publications.

The theoretical basis was formed by studies that examine different aspects of applying LLMs to defect detection and resolution tasks. The work of Alhanahnah M. [6] conducted an empirical evaluation of the effectiveness of pre-trained models in repairing declarative specifications, where the role of agents and auto-prompts was tested. The study by Ardimento P. [3] proposed an LLM-based classifier capable of predicting the time to fix bugs in defect tracking systems, which expands the possibilities for managing test cycles. Of particular importance for the methodological part are the works of Chen Y. [4], which describes an approach to repairing flaky tests using LLMs, and the work of Chen Y. [5], which presents a model for the automatic analysis of incident root causes in cloud systems. These studies showed that modern language models can be considered both a tool for finding errors and a means for their interpretation and resolution.

Boffa M. [2] and Qi J. [10] made a valuable contribution to the systematization of data on the use of LLMs in log analysis. The former proposed LogPrécis—a methodology for the automated analysis of malicious logs, while the latter developed LogGPT, for the first time applying ChatGPT for anomaly detection in system journals. Their approaches formed the basis for comparing classical log processing methods with new LLM-oriented solutions. The work of Cui T. [7] created a large-scale test environment, LogEval, which allows for the objective comparison of the performance of language models in log analysis, serving as an additional basis for our analysis.

The empirical part of the review was supplemented by the developments of Sun Y. [11], who proposed SemiSMAC—a semi-supervised system for anomaly detection with automatic hyperparameter tuning, and the study by Guan W. [8], which created the LogLLM architecture, combining traditional log analysis with the capabilities of LLMs. An additional direction was considered in the work of Dakhama A. [7], which shows how language models enhance error detection methods in system simulators.

The methodological structure of the study is built on a multi-dimensional comparison, from the repair of formal specifications to log analysis and the resolution of flaky tests. For comparison with academic solutions, an industrial installation was used, which combines log

analysis and test run management in a single CI/CD loop. The architecture includes an MCP server for request orchestration and results caching, Amazon Q CLI for forming queries to the logs, and Claude 4 for the semantic grouping of errors and analysis of failure causes. The integration is implemented on top of an existing pipeline (Cypress, Allure), which allowed for the centralization of incident processing and a reduction in tool fragmentation. The system is deployed on "nightly" runs and generates reports with prioritization and aggregated "error templates" suitable for further retrospectives and auto-classification.

Results

An analysis of modern approaches to using language models in automated testing shows that researchers formulate tasks and build architectures in different ways. The study by Chen Y. [5] presents the RCACopilot system, which performs root cause analysis of incidents. It uses summaries of diagnostic information, and the GPT-4 model itself is used in a few-shot learning mode with elements of step-by-step reasoning. This application demonstrates that a language model can perform the role of a classifier for incident categories. The work by Guan W. [8] proposes the LogLLM architecture. It is built on a combination of an encoder and a decoder: input log sequences undergo normalization, then a multi-stage fine-tuning scheme is used, and to reduce resource intensity, optimization with a reduced computational volume is applied. The study by Chen Y. [4] describes the FlakyDoctor method, which is designed to repair flaky tests. The system is implemented as an iterative process: the language model generates a fix, and a built-in validator checks its correctness and, if necessary, triggers a re-generation. The study by Qi J. [10] developed LogGPT, where ChatGPT is used for anomaly detection in logs. Different options for representing the input data are used—from raw messages to cleaned and aggregated sequences. The task is formulated in the form of a prompt, and the model makes a decision about normality or anomaly and provides an explanation for the result.

The solutions under review demonstrate a wide range of directions. Some are focused on identifying the root causes of incidents, others on detecting anomalies in logs, and still others on repairing tests or interpreting detected failures. The role of the language model in these systems varies. In some cases, it acts as a classifier; in others, it generates fixes or explanations. The preparation of input data also differs. For incident analysis, a brief summary of diagnostic information is used; for logs, normalization of sequences is applied; and in testing, parsing and failure localization are important. The training methods also differ. Some solutions are based on using a small number of examples with step-by-step reasoning, while others require additional model training or combined architectures that unite encoders and decoders. Differences are also observed in the target outputs. Some works record the root cause category, others determine a binary distinction between normal and anomalous behavior, and in studies on testing, the result is a decision on whether the test was successfully repaired. A structured comparison is presented in Table 1.

Table 1 – Architectures and tasks of LLM-based approaches (Compiled by the author based on sources: [7, 8, 9, 10])

Method	Input	Model / training	Output task
RCACopilot	Summaries from diagnostic handlers, k-NN with FastText	GPT-4, training on a small number of examples with step-by-step reasoning	Root cause category (Micro-F1/Macro-F1)
LogLLM	Log sequences (RE normalization)	BERT → projector → Llama; three-stage additional training; QLoRA	Binary “normal / anomaly”
FlakyDoctor	Test runs, traces, failure localization	GPT-4, iterative repair with validation	“Repaired / not repaired” by OD/ID/NOD classes
LogGPT	Raw / content / event log	ChatGPT, prompt-based	“Normal / anomaly”

	sequences	formulation, window, JSON format	with explanation

The comparison shows that the approaches differ in their technical implementation and the concept of applying language models. RCACopilot demonstrates the capabilities of classifying root causes based on diagnostic information [7]. LogLLM shows the effectiveness of hybrid schemes that combine an encoder and a generative decoder [8]. FlakyDoctor reveals the potential of iterative strategies for repairing flaky tests [9]. LogGPT confirms that even without additional training, with a correctly formulated task, it is possible to successfully detect anomalies and generate explanations [10]. The analysis confirms that there is no universal solution. Effectiveness directly depends on the quality of data preparation, the choice of architecture, and the training methods. Language models prove successful in various tasks, but their application requires adaptation to the specifics of particular testing scenarios.

An analysis of data published in peer-reviewed sources demonstrates differences in the effectiveness and application conditions of language models. The study by Chen Y. [5] presents the RCACopilot system, designed for root cause analysis of incidents in cloud systems. Experiments were conducted on a sample of 653 incidents, and the use of GPT-4 achieved a Micro-F1 of 0.766 and a Macro-F1 of 0.533. The average inference time was 4.205 seconds, which reflects a balance between classification quality and computational load. The work by Guan W. [8] describes the LogLLM architecture, oriented towards detecting anomalies in logs. The results were obtained on four datasets: HDFS (F1 = 0.997), BGL (F1 = 0.916), Liberty (F1 = 0.958), and Thunderbird (F1 = 0.966). Additionally, cases of exceeding memory limits on large samples were recorded, which underscores the scalability limitations and points to the need for optimization in industrial implementation. The study by Chen Y. [4] examined the FlakyDoctor method, aimed at repairing flaky tests. For tests of the OD-Victim category, the repair success rate was 78%; for OD-Brittle, it was 51%. For ID-type tests, the overall rate was 58%. A separate comparison on the DexFix dataset showed the advantage of FlakyDoctor (55% successful repairs versus 46% for the original DexFix method). Practical approbation was conducted in open-source projects: developers submitted 61 pull requests with repaired tests, of which 19 were accepted. This confirms the applied value of the method beyond laboratory experiments.

The study by Qi J. [10] presents the LogGPT system, which uses ChatGPT for log analysis. With a configuration of window=50 and the second prompt scheme on the BGL dataset, an F1 score of 0.618 was achieved with a recall of 1.000 and a specificity of 0.087. On the Spirit dataset in few-shot mode, the F1 score was 0.694 with the same recall of 1.000 and a specificity of 0.348. These results show that the model can successfully detect anomalies even without fine-tuning, but the high recall is accompanied by an increase in the number of false positive classifications. Table 2 examines the relationship between the metrics and experimental conditions for all four approaches.

Table 2 – Summary of metrics and experimental settings (Compiled by the author based on sources: [4, 5, 8, 10])

Method	Dataset / condition	Reported metrics
RCACopilot	653 incidents; GPT-4	Micro-F1 0.766; Macro-F1 0.533; inference 4.205 s
LogLLM	HDFS	F1 0.997
	BGL	F1 0.916
	Liberty	F1 0.958

	Thunderbird	F1 0.966
FlakyDoctor	OD-Victim	78% repaired
	OD-Brittle	51%
	ID total	58%; DexFix set: 55% vs 46% (DexFix)
LogGPT	BGL, window=50, Prompt-2/few-shot	F1 0.618; Recall 1.000; Specificity 0.087
	Spirit, window=50, Prompt-2/few-shot	F1 0.694; Recall 1.000; Specificity 0.348

A comparative analysis shows that RCACopilot demonstrates stable results in root cause classification tasks with a moderate load [5]. LogLLM achieves high accuracy on various datasets but is accompanied by resource limitations [1]. FlakyDoctor provides a noticeable improvement in test repair compared to previous approaches [2]. LogGPT confirms the applicability of universal language models to the task of log analysis, although a significant number of false positive classifications are observed [11]. The performance indicators, in aggregate, point to the high potential of integrating language models into testing processes, but at the same time, they record the presence of limitations that require the adaptation of solutions to specific application conditions.

Discussion

The analysis of the sources conducted shows that the choice of method for working with errors in automated tests directly depends on the type of task and the application conditions. Different architectures of language models produce results only when considering the specifics of the input data and the goals of the analysis. For root cause analysis of incidents, the most appropriate approach is to use a pipeline that includes processing signals from different diagnostic sources, summarizing them, and then categorizing them with a language model. This approach is implemented in the study by Chen Y. [5], where RCACopilot showed the ability to combine different types of data and to classify root cause categories with high accuracy. The application of such a solution is justified in large infrastructure projects where the volume of information from logs, traces, and monitoring systems exceeds the capabilities of traditional manual analysis.

When working with long log sequences and unstable patterns, a hybrid-type architecture that combines an encoder and a decoder proves to be more effective. The efficiency of the hybrid scheme observed in operational data confirms the conclusions about the advisability of separating the representation and decoding functions. In a production environment, the role of a "lightweight" embedder and router is performed by the MCP server and Amazon Q CLI, while the interpretation and summarization functions are handled by Claude 4. Such a separate loop reduces the load on central computing nodes and simplifies scaling by source type (journals, traces, report artifacts), while maintaining the quality of grouping and prioritization. The study by Guan W. [8] demonstrated that LogLLM can achieve almost perfect accuracy on different datasets by using memory optimization through an embedder and a projector. This result confirms that when processing large arrays of logs, the combination of sequence representation and decoding mechanisms is critically important, as it reduces the load on computational resources and preserves the quality of the analysis. The practical value of such an approach is that it is applicable in systems with a large number of similar events, where it is necessary to quickly separate normal processes from anomalies.

For test errors of types OD and ID, a promising direction is the application of iterative test repair with verification of the fixes. The study by Chen Y. [4] showed that FlakyDoctor successfully repairs more than half of flaky tests. This approach demonstrates a significant advantage over previous methods and is particularly useful in projects with long regression testing cycles, where the time to fix errors directly affects the release of new product versions. However, for tests of type

NOD, the results remain weak. Automation does not yet allow for reliable fixes, which indicates the need for additional research.

When a quick check and explanation of what is happening in the logs "on the fly" is required, a possible solution is to use the method of formulating the task through a prompt. In operational use, the prioritization of test runs is implemented through an analysis of the history of failures and commit descriptions in natural language. Claude 4 identifies related risk areas, and the MCP server forms a shortened list of runs. This reduces the load on the test stands during peak hours and provides faster feedback to developers without degrading defect penetration rates.

The study by Qi J. [10] showed that LogGPT can find anomalies without fine-tuning, providing high recall rates, but at the cost of sacrificing specificity. This approach can be used for rapid analysis in systems where speed is important and the ability to get an explanation is valuable, even if the risk of false positive classifications increases.

A comparison of these methods allows us to assert that each solution should be selected for a specific scenario. For root cause analysis, pipelines with pre-processing and categorization are suitable. For large arrays of logs, hybrid architectures with resource optimization are effective. For repairing automated tests, the iterative process with validation of fixes shows the greatest effect. For quick and explainable answers, the scenario of using prompts remains in demand. In aggregate, this forms a holistic understanding of the boundaries and capabilities of modern language models in software quality management.

The analysis of the results allows for the identification of a number of limitations that affect the possibility of applying modern language models in practical conditions. The key risks and limitations are summarized in Table 3.

Table 3 – Limitations and risks across studies (Compiled by the author based on sources: [4, 5, 8, 10])

Method	Limitations (from sources)
RCACopilot	Dependence on monitor triggers and availability of handlers; limited applicability without a detector; variability of LLM responses; transferability between services remains an open issue
LogLLM	Out-of-memory errors when feeding long sequences into Llama; high GPU and training time requirements; reliance on labeled data for supervision; sensitivity to window size and configuration
FlakyDoctor	NOD-flaky tests remain largely unrepaired; costly reproduction and validation; risk of “fixes at any cost” (e.g., removing assertions), which require oversight
LogGPT	High rate of false positives; sensitivity to prompts and window size; risk of hallucinations and unreliable outputs; constraints from response length limits

The limitations presented in Table 3 are directly reflected in the practical use of the described methods. For RCACopilot, the key barrier is the dependence on the completeness of the data coming from monitors and handlers [2]. If the diagnostic infrastructure is not fully deployed, the model does not receive enough information for the correct categorization of root causes.

In the case of LogLLM, the main difficulty lies in its resource intensity [8]. High accuracy is achieved by processing long log sequences, but direct feeding of data into Llama results in out-of-memory errors. This makes the model sensitive to the volume and form of input data and increases the requirements for graphics accelerators. At the same time, a dependence on the availability of labeled samples remains, which complicates implementation in companies without pre-prepared datasets. FlakyDoctor showed a high result on tests of types OD and ID, but it was not possible to completely cope with NOD tests [4]. An additional problem is the high cost of reproducing and validating such errors. The risk of incorrect fixes is noted separately, where the model eliminates a failure by removing important checks, which can lead to a

decrease in trust in the system. LogGPT demonstrated its value for the rapid analysis of logs, but the results are accompanied by a high rate of false positive classifications [10]. The sensitivity to the wording of prompts and the choice of a window confirms that the model remains dependent on engineering decisions in the area of prompts. An additional limitation is the response length limits, which make it difficult to interpret the results with large volumes of data.

The combination of these factors shows that the practical implementation of the considered solutions requires not the autonomous launch of models, but their integration into existing pipelines. Filters, threshold values, and procedures for validating patches are necessary to reduce the risk of false alarms and errors in automatic fixes. In addition, the high sensitivity of the models to parameters indicates the importance of MLOps practices, where issues of calibration, control of computational resources, and verification of the reliability of results should be considered as part of the standard operation process.

Conclusion

The study conducted has allowed for the systematization of modern methods for analyzing and classifying errors in automated tests using language models. The approaches considered demonstrated a variety of architectural solutions, data preparation methods, and training strategies, which made it possible to identify their strengths and weaknesses. It has been established that the integration of models into testing processes provides an expansion of the capabilities for diagnosing and repairing errors, but the effectiveness of their application directly depends on the type of tasks and the operating conditions.

The analysis of academic sources confirmed the importance of using pipelines with data pre-processing for the classification of incident root causes and the effectiveness of hybrid architectures when working with long and unstable log sequences. In the area of test repair, the effectiveness of iterative fixing with mandatory validation of changes was shown, which is particularly important for the stability of regression runs. Alongside this, it was established that a priority on rapid response can be ensured by methods based on formulating tasks through prompts, although such solutions are accompanied by an increase in the number of false alarms.

The systematization of risks showed that the limitations for practical implementation remain the dependence on monitoring infrastructure, requirements for computational resources, the complexity of reproducing flaky tests, and high sensitivity to prompt parameters. These factors indicate the need to integrate methods not in isolation, but as part of comprehensive pipelines where filtering, threshold mechanisms, and validation procedures are implemented. Such an approach allows for minimizing operational costs and reducing the risk of unreliable results.

Of particular importance is the consideration of practical experience, which confirms that the implementation of AI models can significantly increase the stability of nightly runs, reduce the overall time for regression testing, and decrease the load on QA teams. Practice has shown that the combination of an MCP server, Amazon Q CLI, and Claude 4 is transferable between teams with minimal adaptation. A unified orchestration loop is maintained, and the settings for log sources and report templates are parameterized at the integration stage. This facilitates scaling within the organization and reduces the time to bring new projects up to the quality standard.

Thus, language models can be considered a promising tool for improving the quality of automated testing, but their use requires adaptation to specific scenarios and the development of a supporting infrastructure. Prospects for further research are related to the in-depth development of approaches to repairing flaky tests of the NOD category, improving methods for reducing false positive classifications in log analysis, and integrating MLOps practices to ensure the sustainable application of language models in scalable industrial systems.

References

1. Alhanahnah, M., Hasan, M. R., Xu, L., et al. (2025). An empirical evaluation of pre-trained large language models for repairing declarative formal specifications. Empirical Software Engineering, 30, 149. https://doi.org/10.1007/s10664-025-10687-1
2. Ardimento, P., Capuzzimati, M., Casalino, G., Schicchi, D., & Taibi, D. (2025). A novel LLM-based classifier for predicting bug-fixing time in bug tracking systems. Journal of Systems and Software, 230, 112569. https://doi.org/10.1016/j.jss.2025.112569
3. Boffa, M., Drago, I., Mellia, M., Vassio, L., Giordano, D., Valentim, R., & Ben Houidi, Z. (2024). LogPrécis: Unleashing language models for automated malicious log analysis: Précis: A concise summary of essential points, statements, or facts. Computers &

Security, 141, 103805. https://doi.org/10.1016/j.cose.2024.103805

4. Chen, Y. (2024, May 23). Flakiness repair in the era of large language models. In ICSE-Companion '24: Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (pp. 441–443). ACM. https://doi.org/10.1145/3639478.3641227
5. Chen, Y., Xie, H., Ma, M., Kang, Y., Gao, X., Shi, L., Cao, Y., Gao, X., Fan, H., Wen, M., & others. (2024, April 22). Automatic root cause analysis via large language models for cloud incidents. In EuroSys '24: Proceedings of the Nineteenth European Conference on Computer Systems (pp. 674–688). ACM. https://doi.org/10.1145/3627703.3629553
6. Cui, T., Ma, S., Chen, Z., Xiao, T., Tao, S., Liu, Y., Zhang, S., Lin, D., Liu, C., Cai, Y., Meng, W., Sun, Y., & Pei, D. (2024). LogEval: A comprehensive benchmark suite for large language models in log analysis. arXiv. https://doi.org/10.48550/arXiv.2407.01896
7. Dakhama, A., Even-Mendoza, K., Langdon, W., et al. (2025). Enhancing search-based testing with LLMs for finding bugs in system simulators. Automated Software Engineering, 32, 63. https://doi.org/10.1007/s10515-025-00531-7
8. Guan, W., Cao, J., Qian, S., Gao, J., & Ouyang, C. (2025). LogLLM: Log-based anomaly detection using large language models. arXiv. https://doi.org/10.48550/arXiv.2411.08561
9. Kang, S., Chen, B., Yoo, S., et al. (2025). Explainable automated debugging via large language model-driven scientific debugging. Empirical Software Engineering, 30, 45. https://doi.org/10.1007/s10664-024-10594-x
10. Qi, J., Huang, S., Luan, Z., Fung, C., Yang, H., & Qian, D. (2023). LogGPT: Exploring ChatGPT for log-based anomaly detection. arXiv. https://doi.org/10.48550/arXiv.2309.01189
11. Sun, Y., Keung, J. W., Yang, Z., Liu, S., & Liao, Y. (2025). SemiSMAC: A semi-supervised framework for log anomaly detection with automated hyperparameter tuning. Information and Software Technology, 187, 107869. https://doi.org/10.1016/j.infsof.2025.107869.