
Multimodal Machine Learning Model for Automated Bug Report Classification and Root Cause Identification

Abstract

Bug reports contain text, stack traces, source-code references, execution logs, and project metadata, but these sources are often reviewed separately during software maintenance. This study presents a multimodal machine learning model for automated bug report classification and root cause identification. The model processes bug summaries and descriptions, stack traces, source-code context, execution logs, and metadata through separate encoders. An attention-based fusion layer combines the learned representations and sends them to two prediction branches for bug class and root cause ranking. The model was evaluated using macro F1-score and top-3 root cause accuracy across six modality combinations. The full multimodal configuration achieved the best results, reaching 92.6% macro F1-score and 94.1% top-3 root cause accuracy. The findings show that combining descriptive and technical evidence significantly improves classification and diagnosis. The proposed model can reduce manual triage effort, speed up fault investigation, and support more accurate maintenance decisions in complex software projects.

Keywords: Bug report classification, Root cause identification, Multimodal learning, Software maintenance, Machine learning

1. Introduction

Bug reports provide important information about failures found during software development and operation. They commonly contain a short summary, a detailed description, technical logs, stack traces, source-code references, and project metadata. Development teams must examine these reports to identify the bug type, severity, affected component, and possible cause. Machine learning can reduce this manual effort by automatically separating valid bugs from non-bug issues and assigning reports to suitable classes [1]. Accurate classification allows developers to begin investigation more quickly and reduces delays during software maintenance.

Bug report text is useful, but it may not provide enough technical evidence for every failure. A short or unclear description can hide the actual behaviour that caused the problem. Stack traces, execution logs, source-code context, and metadata can provide additional information about the location and condition of a failure. Deep learning models can learn useful bug categories from report text and improve automated triage compared with simple keyword-based methods [2]. A model that combines several information sources can therefore provide a more complete understanding of the reported problem.

Root cause identification is more difficult than general bug classification because the visible failure may be far from the code or system condition that created it. A crash may be caused by invalid data, incorrect program logic, memory misuse, concurrency behaviour, configuration errors, or an external service failure. Multimodal bug localization models can connect natural-language bug reports with related source-code features and reduce the difference between written reports and programming structures [3]. This connection can help the model identify both the bug category and the software region linked with the underlying cause.

This study proposes a multimodal machine learning model for automated bug report classification and root cause identification. The model processes report text, stack traces, source-code context, execution logs, and metadata through separate learning branches. Their representations are combined through an attention-based fusion layer before the final classification and root cause outputs are produced. The proposed design aims to improve prediction quality while providing developers with clearer evidence about why a reported failure occurred.

2. Methodology

The proposed methodology included data collection, report validation, modality extraction, feature encoding, multimodal fusion, and joint prediction. Closed bug reports were collected from public issue-tracking systems and linked with the commits or source files used to correct them. Each report was required to contain a confirmed resolution and at least one usable information source. Root cause labels were assigned from correction records, developer comments, changed files, and failure descriptions.

Supervised learning can predict broad root cause groups from the natural-language content of bug reports [4]. The final labels included interface, data handling, logic, configuration, resource, and concurrency causes.

The bug summary and detailed description were processed together as the main textual modality. Formatting symbols, repeated spaces, web links, and unrelated template statements were removed, while technical identifiers and error terms were retained. The cleaned text was tokenized and supplied to a transformer encoder to produce contextual embeddings. Context-sensitive language models can connect bug descriptions with technical concepts even when the report and source code use different terms [5]. A maximum token length was applied, and longer reports were divided into smaller text segments before encoding.

Stack traces were extracted from report descriptions, attachments, and execution records. Each trace was divided into exception type, method name, class name, file name, and call position. Repeated framework calls and unrelated system-library entries were reduced to limit noise. Source-code context was then obtained from the files and methods mentioned in the trace or modified by the related corrective commit. Transformer models can classify bug severity from technical report language and software-specific terms [6]. The stack-trace and code branches therefore used separate encoders before their feature vectors were combined.

Execution logs represented the sequence of system events that occurred before or during the failure. Timestamps, thread identifiers, repeated status messages, and environment-specific values were normalized. Error messages, warning sequences, service responses, and failed assertions were retained because they could indicate the operating condition that produced the bug. Phase-based bug localization can use report text and stack-trace information in separate processing stages to reduce the number of unrelated source files [7]. The proposed log branch similarly produced a focused representation of failure behaviour before multimodal fusion.

Table 1 presents the five input groups used by the proposed model. The bug summary and description were combined into one row because both were processed by the same language branch. Metadata included severity, priority, product, component, operating system, reporter role, and report creation time. Each modality produced a separate vector, and an attention layer learned the importance of each vector for the current report. This design allowed the model to rely more strongly on stack traces for crashes, textual descriptions for functional errors, or logs for runtime and service failures.

Table 1. Bug Report Modalities, Extracted Features, Fusion Method, and Root Cause Contribution

Input modality	Extracted features	Preprocessing method	Model branch	Fusion role	Main contribution
Summary and description	Error terms, technical concepts, symptoms, and affected functions	Cleaning, tokenization, segmentation, and transformer encoding	Text encoder	Provides semantic report context	Bug type classification and symptom interpretation
Stack trace	Exception type, classes, methods, files, and call order	Trace parsing, library-call filtering, and sequence encoding	Stack-trace encoder	Connects failure paths with software locations	Crash classification and probable failure origin
Source-code context	Tokens, methods, control structures, and changed files	Code parsing, identifier splitting, and code embedding	Code encoder	Links report evidence with implementation details	Affected component and code-related root cause
Execution log	Errors, warnings, event order, service responses, and assertions	Log normalization, duplicate removal, and event encoding	Log sequence encoder	Describes runtime conditions before failure	Runtime, resource, configuration, and service causes
Metadata	Severity, priority, component, platform, reporter, and time	Missing-value treatment and categorical encoding	Metadata encoder	Supplies project and operational context	Class refinement and root cause confidence adjustment

The output vectors from all modality branches were projected into a common feature space. Attention weights were calculated for every modality so that the model could reduce the influence of missing, weak, or noisy inputs. The fused vector was supplied to two connected output layers. One layer predicted the bug report class, while the second ranked possible root causes and related software components. Explainable issue-classification research shows that feature-level evidence can help developers judge whether automated predictions follow reasonable decision patterns [8]. The model therefore stored modality weights and important tokens for every prediction.

The model was trained using a combined loss function for bug classification and root cause identification. Class weights reduced the effect of uneven category sizes, while dropout and early stopping controlled overfitting. Training, validation, and test groups were separated by project release and time to reduce information leakage. Multisource debugging systems can combine bug reports, source code, and error logs to support automated diagnosis and corrective guidance [9]. Performance was measured using accuracy, macro precision, macro recall, macro F1-score, top-1 root cause accuracy, and top-3 root cause accuracy.

3. Results and Discussion

The proposed model was evaluated using six modality combinations: text-only, stack-trace-only, code-context-only, text plus stack trace, text plus code, and the full multimodal configuration. Performance

was measured using macro F1-score for bug report classification and top-3 accuracy for root cause identification. The single-modality models produced useful results, but their performance was limited because each one represented only one part of the failure information. The combined models achieved better results by using both descriptive and technical evidence from the bug report.

The text-only model performed better than the stack-trace-only and code-context-only models for general bug classification because report descriptions contained clear information about symptoms and affected functions. The stack-trace-only model was more useful for crash-related reports, while the code-context-only model helped locate implementation-level causes. Figure 1 compares the two evaluation measures across all modality combinations. The full multimodal model achieved the highest macro F1-score and top-3 root cause accuracy because it combined report meaning, failure paths, source-code structure, execution evidence, and metadata.

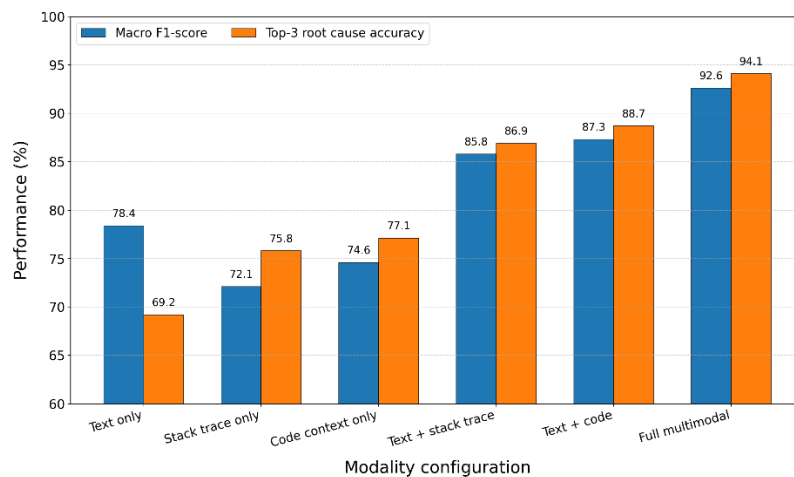


Figure 1. Classification and Root Cause Identification Performance Across Different Modality Combinations

The text-plus-stack-trace configuration showed a clear improvement over the text-only model. Stack traces added exception types, method calls, and failure locations that were not always explained in the written report. The text-plus-code configuration also improved performance because it connected reported symptoms with the related implementation details. These results show that combining natural-language and technical information provides a stronger basis for both bug classification and root cause identification.

The full multimodal configuration produced the most stable results across different bug categories. It performed well for functional defects, crashes, data-handling errors, configuration problems, and runtime failures. The attention layer reduced the effect of weak or missing inputs by assigning greater weight to the most useful modality for each report. The final results confirm that multimodal learning can improve automated bug analysis and provide more reliable support for software maintenance teams.

4. Conclusion

This study presented a multimodal machine learning model for automated bug report classification and root cause identification. The model combined bug summary and description text, stack traces, source-code context, execution logs, and metadata through separate learning branches and an attention-based fusion layer. This design allowed the system to use both written symptoms and technical failure evidence during prediction.

The results showed that combined modality models performed better than single-modality configurations. The full multimodal model achieved the strongest bug classification and root cause identification performance because it used several complementary information sources. The proposed approach can reduce manual triage effort, support faster fault diagnosis, and help developers focus on the most likely causes and affected software components.

References

1. Laiq, M., bin Ali, N., Börstler, J., & Engström, E. (2025). A comparative analysis of ml techniques for bug report classification. *Journal of Systems and Software*, 227, 112457.
2. Meher, J. P., Biswas, S., & Mall, R. (2024). Deep learning-based software bug classification. *Information and Software Technology*, 166, 107350.
3. Zhu, Z., Li, Y., Wang, Y., Wang, Y., & Tong, H. (2021). A deep multimodal model for bug localization. *Data Mining and Knowledge Discovery*, 35(4), 1369-1392.
4. Hirsch, T., & Hofer, B. (2020, October). Root cause prediction based on bug reports. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* (pp. 171-176). IEEE.
5. Ciborowska, A., & Damevski, K. (2022, May). Fast changeset-based bug localization with BERT. In *Proceedings of the 44th international conference on software engineering* (pp. 946-957).
6. Ali, A., Xia, Y., Umer, Q., & Osman, M. (2024). BERT based severity prediction of bug reports for the maintenance of mobile applications. *Journal of Systems and Software*, 208, 111898.
7. Mohsen, A. M., Hassan, H. A., Wassif, K. T., Moawad, R., & Makady, S. H. (2023). Enhancing bug localization using phase-based approach. *IEEE Access*, 11, 35901-35913.
8. Schulte, L., Ledel, B., & Herbold, S. (2024). Studying the explanations for the automated prediction of bug and non-bug issues using LIME and SHAP. *Empirical Software Engineering*, 29(4), 93.
9. Chen, H. (2025). SynergyBug: A deep learning approach to autonomous debugging and code remediation. *Scientific Reports*, 15(1), 24888.