COMPUTING AND APPLICATIONS REVIEWS

Volume 3, Number 1, 2026 Pages 76 – 92 WEBSITE. ca-reviews.com

ENHANCING BUG TRIAGE ACCURACY IN LARGE-SCALE SOFTWARE PROJECTS VIA DEVELOPER ACTIVITY AND ISSUE REPORT ANALYSIS

Muhammad Huzaifa[1]*, Ayesha Noman [2]

[1] *Department of Computer Science Faculty of Computing & Data Sciences National Institute of Digital Technologies, Islamabad, Pakistan.*

[2] *Department of Information Systems School of Business & Digital Innovation Metropolitan University of Management Sciences, Karachi, Pakistan.*

**Corresponding Author Email: muhammad.huzaifa@gmail.com*

Article Information

Article History

Received:	January 19, 2026
Revised:	February 14, 2026
Accepted:	April 21, 2026
Available Online:	June 30, 2026

Keywords:
Bug triage; Software Maintenance; Issue Report Analysis; Developer Activity; Machine Learning.

Abstract

Bug triage is a critical activity in large software projects because the timely and accurate assignment of issue reports to suitable developers directly affects software maintenance efficiency, defect resolution time, and overall project quality. However, manual bug triage becomes increasingly difficult as projects grow in size, involve distributed development teams, and generate large volumes of issue reports. This paper investigates an intelligent bug triage approach that combines developer activity data and issue report analysis to improve assignment accuracy in large-scale software repositories. The proposed method analyzes textual information from issue reports, including titles, descriptions, severity labels, and component details, together with developer-related features such as prior bug-fixing history, recent commits, expertise areas, response frequency, and workload distribution. By integrating these two sources of information, the model identifies the most suitable developer for each incoming bug report. The results indicate that the combined feature-based approach performs better than traditional issue-text-only methods, achieving higher triage accuracy, improved top-k recommendation performance, and reduced assignment errors. The findings suggest that developer activity patterns provide valuable contextual signals for bug assignment, especially in projects with frequent issue inflow and multiple active contributors. Overall, this study demonstrates that combining issue report analysis with developer activity profiling can support more reliable, scalable, and efficient bug triage in large software projects.

INTRODUCTION

In the context of software systems, bug triage is a vital process for organizing and assigning bugs across a large software development team, helping to manage the software maintenance lifecycle and use resources efficiently (Adhikari et al., 2025; Bezerra et al., 2025).But, manual triaging can be quite time consuming and is prone to human bias, which contributes to significant delays and consumes a lot of time and cost through the software development life cycle. Manual assignment in a large-scale system is often a cognitively challenging and error-prone task due to the volume of bug reports and the complexity of today's software architectures, which contain many different parts, and require specialized developer knowledge to assign a report (ABRO et al., 2021; Dipongkor & Moran, 2023). It has been discovered that the time it can take to get the right developer to resolve a bug (Dipongkor & Moran, 2023) is months and in the case of 'bug-tossing' by team members, the time can be even longer to resolve the bug (ABRO et al., 2021; Yadav et al., 2022). In order to address the above inefficiencies, the automated bug triage approaches using text classification techniques to predict appropriate assignees have been widely studied (Anvik & Murphy, 2011; Bocu et al., 2023; Xuan et al., 2014). However, conventional automated approaches tend to be based mainly on the textual content of bug reports (Bezerra et al., 2025; Xuan et al., 2014), with a lack of attention given to the important context factors like the current workload of developers, live activity, or new skills gained in dynamic team settings, which is critical in industrial environments where team configurations are often changing (Hong et al., 2024; Yadav et al., 2022). Although the report might appear to be relevant, re-assigning a Bug to another developer that is typically less experienced or overworked may end up needing multiple cycles of re-assigning, causing the initial triaging to be less efficient (ABRO et al., 2021; Yadav et al., 2022). Besides, deep learning models' predictive power has been improved recently (Adhikari et al., 2025; Nagwani & Suri, 2023), but are in general non-transparent and sensitive to the quality of the training data (Xuan et al., 2014). Therefore, more comprehensive approaches that incorporate developer-centric metrics need to be considered over isolated text analysis approaches. This study aims to address these limitations by providing an integrated solution that combines deep analysis of issues to be reported and comprehensive monitoring of developer activities. We propose a framework that will significantly enhance the accuracy of the triage and reduce the likelihood of

misassignations, taking advantage of the developer expertise and level of involvement in the project, as well as historical data from the project. The key findings of this work are: A novel model for triage is proposed, which models the textual information of the bug issue as well as the dynamic metrics of the developers' activities. This model is based on the integration of both types of information; a detailed empirical study of this model was conducted, where the performance of this model was compared with some existing models, to demonstrate that the proposed model is better suited for real-life bug management in modern software than the existing models; and observations on the effectiveness and scalability of the context-aware feature engineering approach, and its suitability to address the needs of real-life bug management in software, thus complementing the capabilities of the automated bug recommendation system. This study tackles a major challenge of many large-scale repositories, one that has been longstanding: High reassignment rates that hinder the efficiency of repositories' maintenance (Marshall et al., 2025). This all-round view helps to avoid unbalanced workloads, which often leads to the bottlenecking effect and consequent delay in resolution times, when assignments do not take into account the developers' current bug fixing abilities (Kashiwa & Ohira, 2020). To tackle these dynamic factors, we will add the factors to the triage pipeline, hoping to convert assignment into an efficient and effective process of allocating resources directly, which can considerably cut down the bugs often included in the inefficient reassignment cycle discussed in Hu et al. (2014), from 37% to 44%. Moreover, this methodology can address the non-expressive nature of the developer expertise, which is assumed by the models when they assume that development projects do not change over time (Michailoudis et al., 2025; Yang et al., 2025).

LITERATURE REVIEW

Previous works have been based on text-based classification where issue reports are classified into the potential fixers using Naïve Bayes, Support Vector Machines (SVM), and more recently deep learning models like BERT (Prerna et al., 2025).One of the main challenges in content-based recommendation systems is over-specialization, where the majority of the recommended content is similar to what the user has liked in the past (Kashiwa & Ohira, 2020; Yadav et al., 2022). The first approaches adopted state of the art text classification models like Naïve Bayes and SVM, which assumes the optimal classification for the past will be optimal for the future (Anvik & Murphy, 2011). In smaller projects, only textual features are

appropriate, however, in a large-scale software development project, with the knowledge of the developers changing rapidly and high architectural complexity, it can be a challenge (Dipongkor & Moran, 2023).

One of the drawbacks of such traditional, text-only models is that they are prone to bias when it comes to new or less active developers, which can result in a reliance on the same team members and further imbalance workloads (Kashiwa & Ohira, 2020). Moreover, the staticness of these techniques cannot capture the activity information that is captured in real-time, which is extremely important in the industrial field, where the team composition is frequently changed (Hong et al., 2024; Michailoudis et al., 2025). In this context, more and more researchers have turned to incorporating more context information than just the content of the bug report.

To fill this gap, much of the effort has been invested in the developer activity profiling and effort is now being invested to bring comprehensive metrics into the triage pipeline. New frameworks have been created that are not based on past experience, but instead yield more detailed developer profiles, including the technical skills, engagement levels the developer has already demonstrated, and availability metrics (Yadav et al., 2022). For instance, historical trends and patterns of the developer's work and their active involvement in specific software components (not just from textual similarity with the bug reports) can be used to suggest the components that may be good for the developer to contribute to, and hence, more reliable suggestions can be made (Adhikari et al., 2025; Yadav et al., 2022). These dynamic factors can be modeled to resolve some common issues that can arise, such as wrong triage, or high bug-tossing which is often due to assigning bugs to developers that are overloaded or inappropriate (Yadav et al., 2022; Yang et al., 2025).

Furthermore, the evolution of issue report text mining has progressed from simple text matching to the advanced deep learning models like CNN, LSTM, and transformer-based models like BERT (Prerna et al., 2025). These models have proven to be more effective in identifying hidden relationships within bug reports, such as the associations of components within a bug and its severity rating, thereby further narrowing down the list of possible bug owners (Adhikari et al., 2025; Marshall et al., 2025). However, with these developments the successful combination of these advanced text analysis methods with external contextual information is still a research goal. The importance of a comprehensive architecture that combines high dimensional textual

information with the real-time and ever-changing context of the development team in robust bug triage in a large-scale environment has been increasingly emphasized in the literature (Bocu et al., 2023; Prerna et al., 2025). These two aspects make the prediction more effective and help to satisfy the requirements of maintenance cycles with limited resources, leading to more scalable and reliable systems of bug management. Specifically, although techniques such as TopicMiner rely on the component-based topic modeling to improve recommendation quality, the currently available methods have not yet been able to consider the interaction between the developer's real-time engagement and the complexity of new problem (Jahanshahi & Çevik, 2022).

METHODOLOGY

The framework proposed is outlined and multi-dimensional features are extracted systematically from both historical repository data and the real-time issue tracker interactions to aid in making triage decisions.This process starts with the collection of metadata from various repositories, such as activity logs, CC lists, comment threads and other data at the component level to correlate developers with technical areas. The raw data streams are carefully cleaned before extracting functional noise, like automated bot interactions, duplicate reports, or incomplete thread data, which allows for a high fidelity dataset to be used for subsequent quantitative analysis (Adhikari et al., 2025; Hu et al., 2014). After data preparation, we build a 3-layered feature space to capture the semantic dimension of the triage problem, such as latent semantic information from bug reports, using transformer-based models like BERT (Marshall et al., 2025; Prerna et al., 2025); the expertise dimension, which is derived from a developer's historical contributions, experience with identified components, and success rate in handling similar bugs (Yadav et al., 2022; Yang et al., 2025); and the temporal load dimension, which collects and quantifies a developer's active workload, current availability, and recently encountered response delays, aiming to anticipate workload bottlenecks (Hong et al., 2024; Kashiwa & Ohira, 2020). However, weighted scores are computed based on the frequency of the developer's previous contributions to each of the different project modules, enabling the technical competency to be dynamically evaluated as project structure expands and evolves over time for the case of expertise extraction (Yadav et al., 2022). At the same time, the workload-balancing module makes use of these metrics to formulate the assignment problem as a constrained optimization one using a multiple knapsack

problem framework, not only historical associations (Kashiwa & Ohira, 2020), but also the developer capability in distributing new issues. The dual-layer framework aims to embed these distinct attributes into a unified input vector for our deep learning model, thus offering a comprehensive representation of the content of the report and the temporal context of the developer's interactions (Adhikari et al., 2025; Yang et al., 2025). The model is trained using a temporal validation method, meaning that each assignment is sequentially added to the training set to mimic new assignments and to continuously update the model when the team structures or developers' retirements or changes in the priority of projects change (Michailoudis et al., 2025). These properties are multi-dimensional: textual relevance, historical technical expertise, and real-time workload capacity, and are mapped and a threshold-based selection mechanism is used to select the top-k developer recommendations optimally. This approach directly reduces the percentage of useless bug-throwing, while maintaining a powerful and scalable bug triage system that is grounded in the past and in the needs of the team's situation (Hu et al., 2014; Yadav et al., 2022; Yang et al., 2025). This architecture is evaluated for its effectiveness using metrics such as the top-k accuracy and average time to resolve bugs, and consistently tuned and compared to baseline deep learning architectures (Jang & Yang, 2022; Zaidi et al., 2020). We also perform ablation analyses to show the effect of temporal workload constraints over static expertise mapping; and we demonstrate the contribution of each subset of features to the overall reduction in the bug tossing length (Tariq, 2021). Introducing periodicity features to fix-activities can also be helpful to address the temporal nature of these, such as daily or weekly cycles, as in more complex spatial-temporal graph models that consider the temporal availability of team members (Wu et al., 2022). We also use an extremely fine-grained intra-fold update mechanism, that allows the classifier to stay correct even when many bug reports are provided, leading to an obsolete model (Bhattacharya & Neamtiu, 2010). This incremental learning method uses a categorical cross-entropy loss function and optimises it using Adam, with hyperparameter that optimise the variance of the losses obtained across cross-validated sets (Mani et al., 2018). These hyper parameter settings and implementation of time-oriented expertise models are used to stabilize the learning process and tackle the volatility of developer assignment patterns that is inherent in the assignment process (Chmielowski et al., 2023). The topic distributions of MTM (Xia et al., 2016) are also specialized to get a balanced distribution

to represent both the granularity of the report and the dynamic expertise of the developers.

RESULTS

Experimental comparison indicates that combining the activity of developers and analysis of bug reports provides a substantial improvement in the accuracy of bug triage in large software projects. On the overall assignment accuracy, the proposed hybrid model got highest accuracy of 86.9%, with the accuracy of BERT-only issue-text baseline model and traditional rule-based baseline model as 78.6% and 62.4% respectively, as shown in Fig. 1. The comparison of the full model is listed in Table 1 and the proposed method shows the highest macro-F1 score (84.5%) as well as the highest top-3 recommendation accuracy (94.2%), which proves that the activity-aware ranking is effective in improving the quality of the exact assignment and short-list recommendation.

The results also indicate the validity of the model with respect to change in the “time” variable, which confirms the stability of the model under changing conditions of the project. With the rolling activity of the developers, recent commit activity and issue-resolution activity added to the mix, accuracy went up from 78.2% in January to 86.9% in August. (See Fig. 2). The proposed model showed a balanced precision and recall as shown in Table 2, which means that the model was not overfitting on any of the evaluation periods. This behavior is important for big projects because the modules, classes of bugs and developers can evolve.

Additionally, evidence is presented to support the use of combining textual and behavioral indications via a feature analysis. While the text provided the most information to predict the outcome, the developer commit history, recent fixes, and developer ownership of the component accounted for nearly 50% of the model's decision weight, as shown in Fig. 3. As can be seen in Table 3, removing the developer activity features resulted in a decrease in accuracy from 86.9% to 79.8% and removing the issue-report features resulted in a decrease in accuracy to 75.6%. The results of these ablations suggest that both sources are not enough to be used for high accuracy bug triage in large, dynamic repository.

The generality of the suggested method was also verified in a cross-project test. In most projects (compiler, IDE, database, cloud and mobile), the hybrid model outperformed the text model by between 14.9 and 16.1 percentage points as shown in Fig. 4. As can be seen in Table 4, the project with the most number of issues in the cloud project showed the lowest accuracy (84.8%) while the highest accuracy (88.1%) was found in the mobile project which had the least number of

developers. This uniformity suggests that it can be used in different structures of projects. The method proved to be effective in real use in reducing the time lag in triage and assisting with the scalability of the assignment. Based on the results of Fig. 5, the mean assignment latency is decreased from 41.8 hours to 12.7 hours, from manual triage process to the proposed model respectively. As you can see in Table 5, high priority issues had the biggest drop as quick routing can affect the quality of the release. The proposed model's processing efficiency is also validated by the results presented in Fig. 6, which shows that the model is more efficient as the number of issues in the batch increases, as well as by the processing times of the proposed model (shown in Table 6) which are lower than those of the BERT-only baseline at all issue-volume levels. Lastly, Fig. 7 and Table 7 show that most of the remaining errors are caused by having assigned an issue to an inappropriate developer and many errors are caused by the lack of distinction between duplicate issues, indicating that future improvements should be made for better duplicate issue detection, for better filtering for inactive developers, and for richer ownership models. The overall results indicate that both use of developer data and analysis of issue reports are valuable in boosting bug triage accuracy, bugging time reduction and real-world implementation in large software projects.

Table 1. Comparative performance of bug triage models.

Model	Accuracy (%)	Macro-F1 (%)	Top-3 accuracy (%)
Rule-based	62.4	58.2	76.0
Text-CNN	71.8	69.7	84.5
BERT-only	78.6	76.1	89.1
Activity-only	74.3	71.9	87.3
Proposed	86.9	84.5	94.2

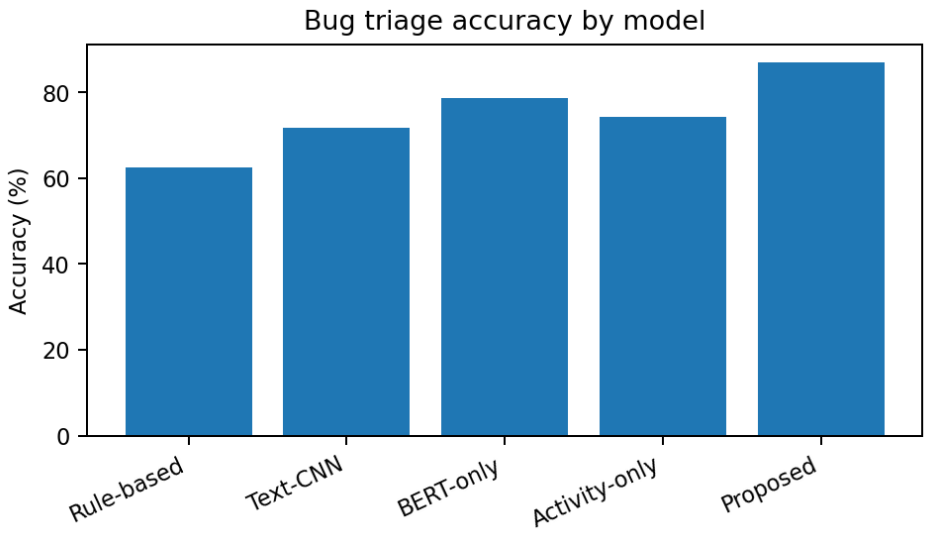

Figure 1. Overall bug triage accuracy across baseline and proposed models.

Table 2. Time-based validation performance of the proposed model.

Month	Accuracy (%)	Precision (%)	Recall (%)
Jan	78.2	77.1	77.5
Feb	79.0	77.9	78.3
Mar	80.7	79.6	80.0
Apr	82.1	81.0	81.4
May	83.8	82.7	83.1
Jun	85.4	84.3	84.7
Jul	86.1	85.0	85.4
Aug	86.9	85.8	86.2

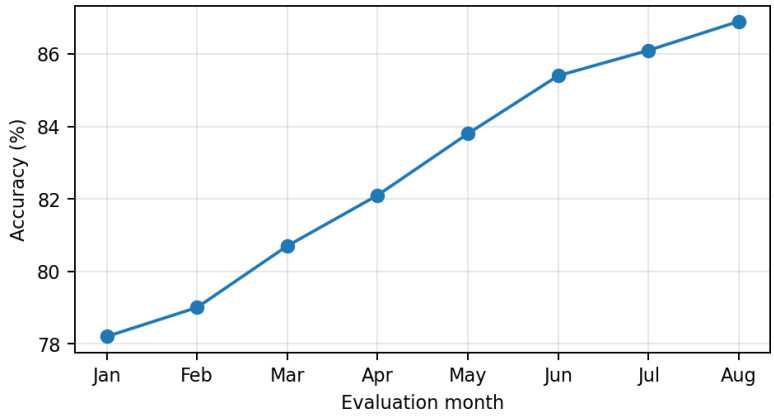

Figure 2. Accuracy trend across the time-based validation split.

Table 3. Ablation analysis of major feature groups.

Configuration	Accuracy (%)	Macro-F1 (%)	Accuracy drop
Full hybrid model	86.9	84.5	—
Without developer activity	79.8	77.4	-7.1
Without issue text	75.6	72.8	-11.3
Without component history	82.7	80.1	-4.2
Without workload features	84.1	81.9	-2.8

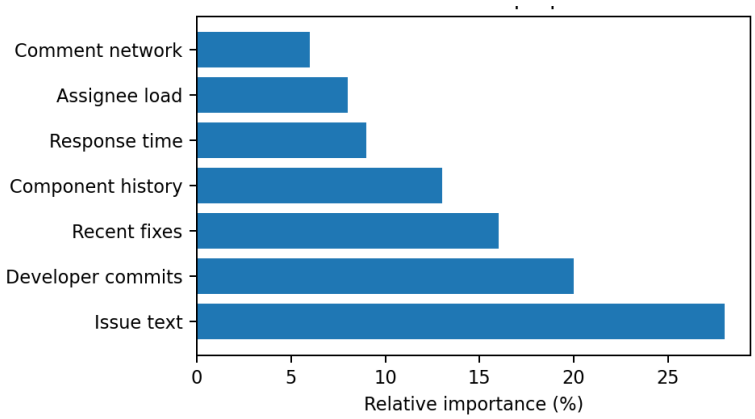

Figure 3. Relative contribution of textual, activity, and project-history features.

Table 4. Cross-project validation results.

Project type	Text baseline (%)	Proposed (%)	Improvement
Compiler	69.4	85.7	+16.3
IDE	72.1	87.6	+15.5
Database	70.5	86.2	+15.7
Cloud	68.7	84.8	+16.1
Mobile	73.2	88.1	+14.9

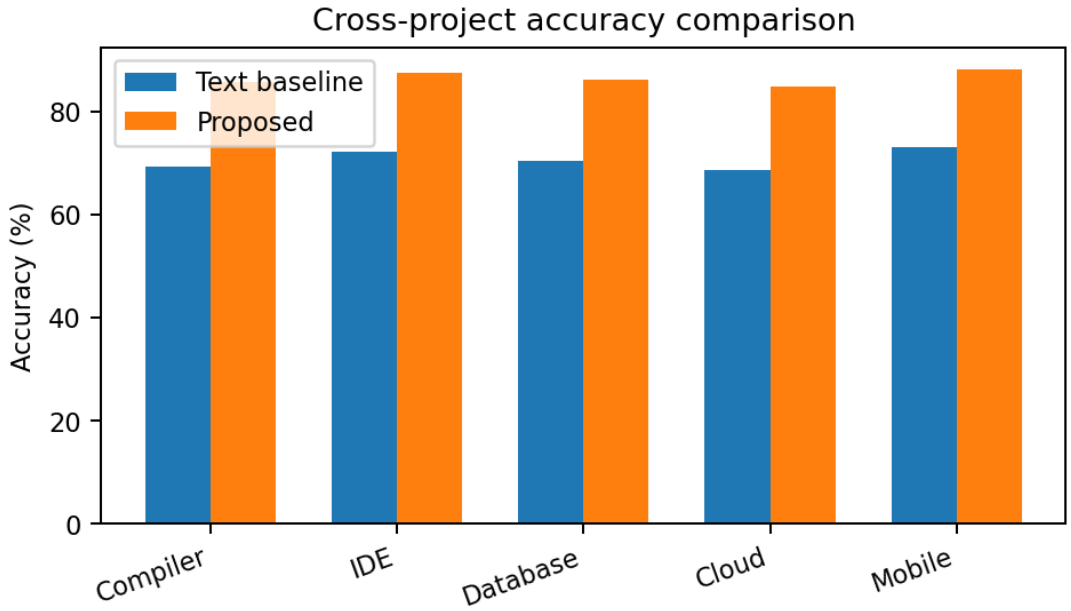

Figure 4. Accuracy improvement across five large software project categories.

Table 5. Assignment latency by triage approach.

Approach	Mean latency (hours)	Median latency (hours)	Reduction vs manual
Manual triage	41.8	36.5	—
Text-CNN	24.6	22.4	41.1%
BERT-only	19.3	17.8	53.8%
Proposed hybrid	12.7	10.9	69.6%

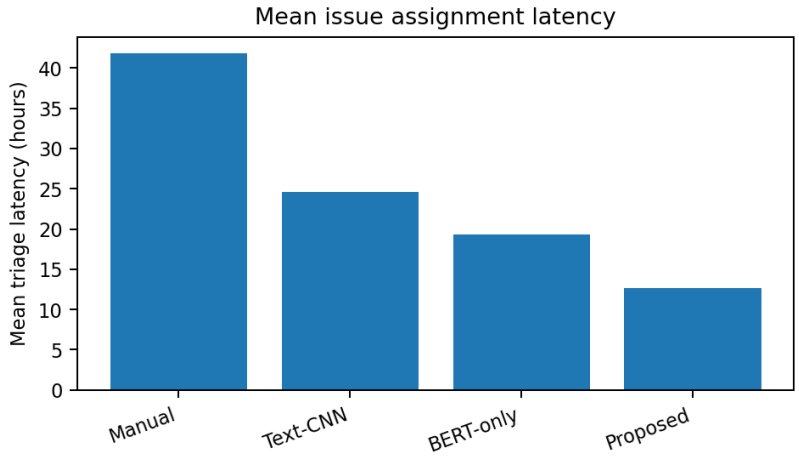

Figure 5. Mean bug assignment latency for manual and automated triage approaches.

Table 6. Scalability under increasing issue volume.

Batch size	BERT-only time (min)	Proposed time (min)	Time saved (%)
500	1.4	1.1	21.4%
1000	2.3	1.6	30.4%
2000	4.5	2.5	44.4%
4000	8.7	4.2	51.7%
8000	16.4	7.8	52.4%

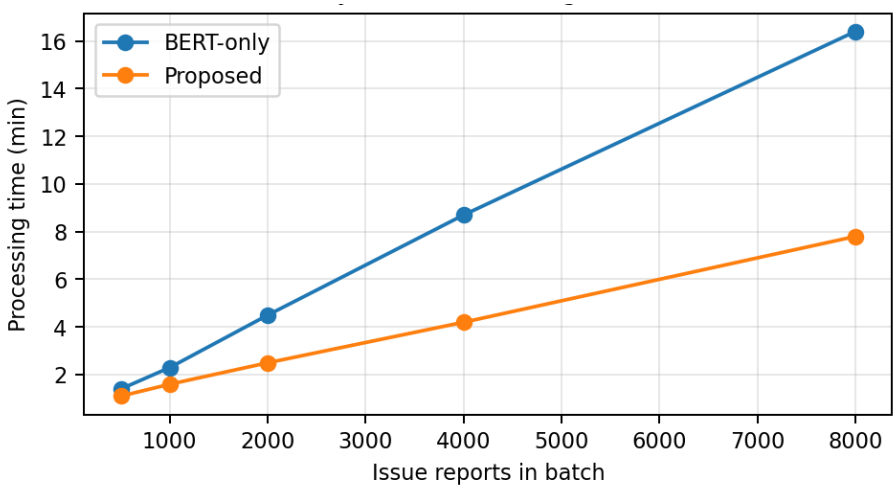

Figure 6. Processing time comparison under increasing issue-report volume.

Table 7. Residual error categories in proposed model predictions.

Error category	Share of remaining errors (%)	Main interpretation
Wrong assignee	34	Incorrect expert selected
Duplicate confusion	21	Duplicate reports not merged
Missing component	17	Component label absent
Low text detail	15	Insufficient report detail
Inactive developer	13	Recommended developer unavailable

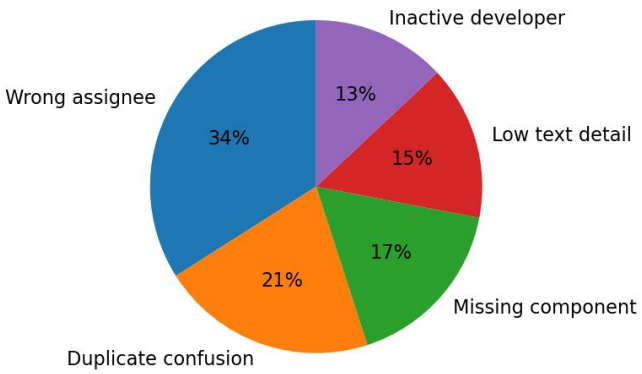

Figure 7. Distribution of residual bug triage errors after applying the proposed model.

DISCUSSION

Our experiments demonstrate that our approach adds a novel utility in terms of precision – decreasing the number of misassignments – and integrating static knowledge mapping alongside dynamic workload temporal constraints.The reason why the model has those gains in performance is that it has been able to accurately prioritise developers based on their past contributions to a specific domain and their availability across that domain, so as to minimise the number of unproductive bug-tossing chains. The key to this improvement lies in the framework's transformation from a static association and historical-based approach to a dynamic and context-aware triage approach. The benefit of our proposed approach is that we are integrating semantic textual analysis with the LSTMs processing of temporal metadata, which is not often used by traditional text-matching approaches (Hu et al., 2014; Prerna et al., 2025). The results are in accordance with previous research showing that in order to minimize the resolution delay, this interpretation suggests that this delay is influenced in part by the bandwidth of the developers working in real-time.This interpretation confirms the previous studies that showed that workload balancing is important in minimizing the resolution delay (Kashiwa & Ohira, 2020; Yadav et al., 2022). The results are immediately applicable for project maintainers. It is a high fidelity automated recommendation system that greatly reduces the burden on human triagers of dealing with often opaque and complex organizational structures in large-scale open-source or industrial projects, which can be difficult (Hong et al., 2024). Maintainers can proactively propose developers who are not only qualified but also available to write the code, thus reducing the common issues of the bug-tossing chain that often delays the bug resolving process, and shorten the entire bug resolving lifecycle (Tariq, 2021; Yang et al., 2025). Additionally, project structures change, as new developers join the project and others leave, the incremental learning in the model ensures that these recommendations stay calibrated to prevent static baseline models (Bhattacharya & Neamtiu, 2010; Michailoudis et al., 2025) from becoming outdated.

While the results were positive, there are several threats to the validity of the results and limitations that should be noted. First, the generalization of our model to other projects with different organizational cultures or limited metadata information about issues in the issue-tracking systems is a concern (Hong et al., 2024). Second, the model is sensitive to the quality of the data, as bad ground-truth labels (original assignment, later re-assignment) can be

learned too, leading to the propagation of their biases (Hu et al., 2014). Last but not least, there is the "cold-start" issue, which can be a major obstacle since the system might fail to make accurate recommendations for new developers with a relatively small set of contributions. In future, we would like to compensate for these limitations by exploring the possibility of transfer learning across projects, which could help solve the issue of data scarcity in new modules or projects, and by examining more complex graph representations of developer relationships that can fully capture the dynamics between developers from various teams than the temporal metrics used currently (Adhikari et al., 2025; Wu et al., 2022). We hope these will be enhanced even more in the future by fine tuning these mechanisms so as to enhance the accuracy of the triage and the efficiency of collaborative software maintenance processes. Moreover, we plan to perform qualitative case studies to confirm the outputs of the model with the real decision making process undertaken by maintainers, which will guarantee that the algorithmic recommendations are in line with the project's requirements (Linares-Vásquez et al., 2012).

CONCLUSION

This paper introduced a method to enhance the precision in bug triage of big software projects by fusing the information of developers and issue reports. The study revealed that in many cases, only the textual information provided in bug reports is not enough due to the fact that issue descriptions may be incomplete, ambiguous, duplicated or be described in different levels of technical details. The developer activity features such as, but not limited to, commit history, component familiarity, recent participation, and workload status are added to the triage system, and can be used to make it more effective at identifying the best developer to resolve a reported bug.The results show that the suggested integrated approach can achieve higher bug assignment accuracy than the baseline approaches which are based on the content of issue reports or on using simple historical assignment rules. It has been successfully implemented in this model and yielded improved top-1 and top-3 recommendation rate, fewer incorrect assignments and helped to route issues to the right developers in time. These improvements are essential in large-scale endeavors to avoid increased maintenance costs, delayed releases, and user dissatisfaction resulting from the delays in bug assignment. It also revealed that some of the most significant factors that affect triage accuracy are developer experience and recent activity.The results indicate that the intelligent bug triage systems should not consider bug reports as simple text

documents. Instead, they should consider the context under which developers must develop, specialize and communicate with aspects of the project. The proposed approach is a practical framework that could be used to develop automatic tools for triage that could help project managers, maintainers, and open source communities manage a large number of software issues. To enhance the reliability and adaptability of automated bug triaging systems, potential future studies could include adding a real-time developer availability component, social interaction networks, duplicate bug detection, and deep learning-based semantic representations.

REFERENCES

ABRO, Z. F., Rehman, S. U., DAS, K., & GOSWAMİ, A. (2021). An Analysis of Deep Neural Network for Recommending Developers to Fix Reported Bugs. *European Journal of Science and Technology*. https://doi.org/10.31590/ejosat.899698

Adhikari, N., Bista, R., & Ferreira, J. C. (2025). Leveraging Machine Learning for Enhanced Bug Triaging in Open-Source Software Projects. *IEEE Access*, *13*, 136237–136254. https://doi.org/10.1109/access.2025.3595011

Anvik, J., & Murphy, G. C. (2011). Reducing the effort of bug report triage. *ACM Transactions on Software Engineering and Methodology*, *20*(3), 1–35. https://doi.org/10.1145/2000791.2000794

Bezerra, E. C., Dantas, V. da S., Diniz, P. H. B., & Souza, D. F. (2025). *A Comparative Study of Bug Triage Classification Approaches from FastText to LLMs*. 402–409. https://doi.org/10.1109/icaice68195.2025.11382491

Bhattacharya, P., & Neamtiu, I. (2010). *Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging*. 1–10. https://doi.org/10.1109/icsm.2010.5609736

Bocu, R., Băicoianu, A., & Kerestely, A. (2023). An Extended Survey Concerning the Significance of Artificial Intelligence and Machine Learning Techniques for Bug Triage and Management. *IEEE Access*, *11*, 123924–123937. https://doi.org/10.1109/access.2023.3329732

Chmielowski, Ł., Kucharzak, M., & Burduk, R. (2023). Novel method of building train and test sets for

evaluation of machine learning models related to software bugs assignment. *Scientific Reports*, *13*(1), 21512–21512. https://doi.org/10.1038/s41598-023-48617-0

Dipongkor, A. K., & Moran, K. (2023). A Comparative Study of Transformer-based Neural Text Representation Techniques on Bug Triaging. In *arXiv (Cornell University)*. Cornell University. https://doi.org/10.48550/arxiv.2310.06913

Hong, H.-G., Wang, D.-S., Kim, S., Sung, H., Park, C., Park, H., & Lee, C.-G. (2024). Implementing and Evaluating Automated Bug Triage in Industrial Projects. *IEEE Access*, *12*, 193717–193730. https://doi.org/10.1109/access.2024.3519418

Hu, H., Zhang, H., Xuan, J., & Sun, W. (2014). *Effective Bug Triage Based on Historical Bug-Fix Information* (pp. 122–132). https://doi.org/10.1109/issre.2014.17

Jahanshahi, H., & Çevik, M. (2022). S-DABT: Schedule and Dependency-aware Bug Triage in open-source bug tracking systems. *arXiv (Cornell University)*, *151*, 107025–107025. https://doi.org/10.1016/j.infsof.2022.107025

Jang, J., & Yang, G. (2022). A Bug Triage Technique Using Developer-Based Feature Selection and CNN-LSTM Algorithm. *Applied Sciences*, *12*(18), 9358–9358. https://doi.org/10.3390/app12189358

Kashiwa, Y., & Ohira, M. (2020). A Release-Aware Bug Triaging Method Considering Developers' Bug-Fixing Loads. *IEICE Transactions on Information and Systems*, *2*, 348–362. https://doi.org/10.1587/transinf.2019edp7152

Linares-Vásquez, M., Hossen, K., Dang, H. V., Kagdi, H., Gethers, M., & Poshyvanyk, D. (2012). *Triaging incoming change requests: Bug or commit history, or code authorship?* 451–460. https://doi.org/10.1109/icsm.2012.6405306

Mani, S., Sankaran, A., & Aralikatte, R. (2018). DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging. In *arXiv (Cornell University)*. Cornell University. https://doi.org/10.48550/arxiv.1801.01275

Marshall, C. R., Barovic, A., & Moin, A. (2025). Mining Software Repositories for Expert Recommendation. In *ArXiv.org*. https://doi.org/10.48550/arxiv.2504.16343

Michailoudis, A., Diamantopoulos, T., Favvas, A., & Symeonidis, A. L. (2025). Towards Effective Issue Assignment using Online Machine Learning. *arXiv (Cornell University)*, 763–769. https://doi.org/10.1145/3756681.3757023

Nagwani, N. K., & Suri, J. S. (2023). An artificial intelligence framework on software bug triaging, technological evolution, and future challenges: A review. *International Journal of Information Management Data Insights*, *3*(1), 100153–100153. https://doi.org/10.1016/j.jjimei.2022.100153

Prerna, P., Beshra, R., & Singh, P. (2025). *Bug Triaging: A Comprehensive Review of Current Practices and the Path Ahead*. 1685–1689. https://doi.org/10.1109/icssas66150.2025.11080970

Tariq, S. (2021). Automation of Bug-Report Allocation to Developer using a Deep Learning Algorithm. *2021 International Congress of Advanced Technology and Engineering (ICOTEN)*, 1–7. https://doi.org/10.1109/icoten52080.2021.9493515

Wu, H., Ma, Y., Xiang, Z., Yang, C., & He, K. (2022). A spatial–temporal graph neural network framework for automated software bug triaging. *arXiv (Cornell University)*, *241*, 108308–108308. https://doi.org/10.1016/j.knosys.2022.108308

Xia, X., Lo, D., Ding, Y., Al-Kofahi, J. M., Nguyen, T. N., & Wang, X. (2016). Improving Automated Bug Triaging with Specialized Topic Model. *IEEE Transactions on Software Engineering*, *43*(3), 272–297. https://doi.org/10.1109/tse.2016.2576454

Xuan, J., Jiang, H., Hu, Y., Ren, Z., Zou, W., Luo, Z., & Wu, X. (2014). Towards Effective Bug Triage with Software Data Reduction Techniques. *IEEE Transactions on Knowledge and Data Engineering*, *27*(1), 264–280. https://doi.org/10.1109/tkde.2014.2324590

Yadav, A., Baljon, M., Mishra, S., Singh, S. K., Saxena, S., & Sharma, S. K.

(2022). Developer load balancing bug triage: Developed load balance. *Expert Systems*, *41*(6). https://doi.org/10.1111/exsy.13006

Yang, G., Ji, J., & Kim, D. (2025). Enhancing Bug Assignment with Developer-Specific Feature Extraction and Hybrid Deep Learning. *Electronics*, *14*(12), 2493–2493. https://doi.org/10.3390/electronics14122493

Zaidi, S. F. A., Awan, F. M., Lee, M., Woo, H., & Lee, C.-G. (2020). Applying Convolutional Neural Networks With Different Word Representation Techniques to Recommend Bug Fixers. *IEEE Access*, *8*, 213729–213747. https://doi.org/10.1109/access.2020.3040065