Automatic Bug Triaging Process: An Enhanced Machine Learning Approach through Large Language Models

Deepshikha Chhabra
Department of Computer Science and Engineering, Chandigarh University, Mohali, Punjab, India
114deepshikha@gmail.com (corresponding author)

Raman Chadha
Department of Computer Science and Engineering, Chandigarh University, Mohali, Punjab, India
dr.ramanchadha@gmail.com

Received: 28 August 2024 | Revised: 27 September 2024 | Accepted: 5 October 2024

Licensed under a CC-BY 4.0 license | Copyright (c) by the authors | DOI: https://doi.org/10.48084/etasr.8829

ABSTRACT

Bug resolution and maintenance are the most critical phases of the software development life cycle. The traditional bug triaging concept refers to the manual assignment of bugs to the appropriate developer after reading the bug details from the bug tracker and further resolving it. The advent of machine learning algorithms provides various solutions for automated bug triaging. Machine learning algorithms can be used to classify bugs and assign each to a developer. Reducing manual efforts optimizes bug-triaging by utilizing manpower in other software development processes. Furthermore, machine learning Large Language Models (LLMs) can be used to take advantage of their natural language processing features and capabilities. This study proposes a machine learning-based embed chain LLM approach for automatic bug triaging. This approach is used to automatically classify bug reports. Based on the results, the appropriate developer is recommended. In addition, the proposed approach is used to automatically predict the priority of bug reports. This paper also discusses the strengths and challenges of the proposed approach.

Keywords-bug triaging; machine learning; large language model; embed chain

I. INTRODUCTION

Software development includes various steps starting from requirement gathering, design, implementation (coding), software testing, and maintenance. Software testing is one of the most crucial phases in which all requirements are tested by feeding the input and comparing the expected and actual outputs. The variation of expected and actual outputs is referred to as a bug. Software malfunctions are referred to as bugs [1]. The bug triaging process consists of evaluating and prioritizing reported software bugs to resolve the most critical and prioritized issues at an early stage. This process helps to ensure that software development teams are addressing the most crucial problems promptly and that end-users can receive fixes for these issues. The steps in bug triaging typically include receiving and studying the bug report, reproducing the issue, checking the priority and severity of the bug, assigning the bug to the appropriate developer, and monitoring the progress to ensure that the issue is fixed promptly. An automated bug-triaging process can increase the efficiency of the process [2-3]. The main objective of bug triaging is to streamline the process of fixing software bugs and improve the overall quality of the software. The bug triaging process starts with the birth of a bug, that is when the tester reports a bug, and then the bugs are assigned to the developer for resolution [4-5]. Once the bugs are resolved, the bug status is changed to resolved. The verification of the bug resolution is performed before the bug is declared to be resolved. The completion of this whole cycle from the new state to the verified state must be completed promptly so that bugs can be fixed at high speed [6-7]. Some bugs have high severity or security-related issues that directly impact business.

A bug report is a formal document that includes information and attributes of the bug, such as title, description, steps to reproduce the bugs, severity, priority, etc. The format of the bug report varies from project to project. The significance of the bug report is to aid the bug-triaging process by prioritizing and fixing the bug efficiently [8-9]. Figure 1 shows the bug lifecycle or the states a bug undergoes throughout the software development process. These are:

- New: The tester reports the bug.
- Assigned: After reading the bug details, the bug is assigned to the appropriate developer.
- In progress: The bug is not resolved yet.
- Fixed/Resolved: Denotes the final resolution of the bug.

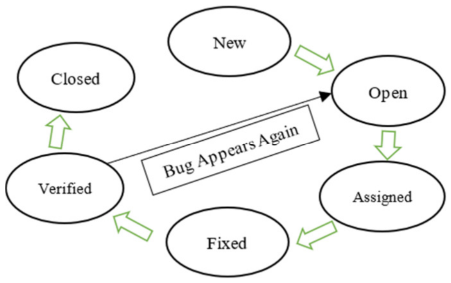
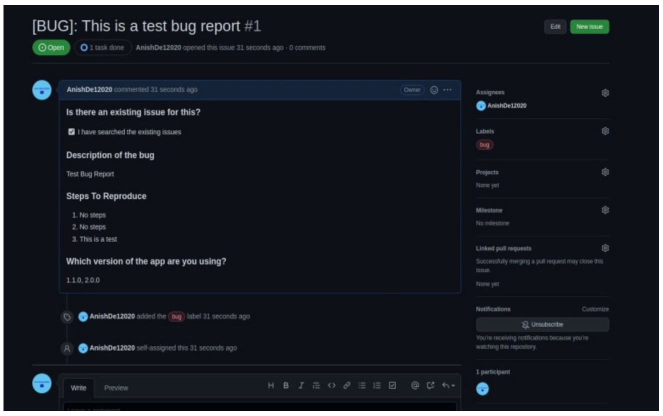


Fig. 1. States of a bug.

Figure 2 demonstrates the bug report template of GitHub's bug tracking tool, which acts as an issue-tracking system to keep a log of all bugs and their statuses. All bugs reported by testers are recorded to manage and keep track of. The most popular bug-tracking software are GitHub, Jam, Bugzilla, and Jira.


[BUG]: This is a test bug report #1

Open · 1 task done · AnishDe12020 opened this issue 31 seconds ago · 0 comments

AnishDe12020 commented 31 seconds ago

Is there an existing issue for this?

- [x] I have searched the existing issues

Description of the bug

Test Bug Report

Steps To Reproduce

1. No steps
2. No steps
3. This is a test

Which version of the app are you using?

1.1.0, 2.0.0

AnishDe12020 added the bug label 31 seconds ago

AnishDe12020 self-assigned this 31 seconds ago

Assignees: AnishDe12020
Labels: bug
Projects: None yet
Milestone: No milestone
Linked pull requests: Successfully merging a pull request may close this issue. None yet
Notifications: Unsubscribe. You're receiving notifications because you're watching this repository.
1 participant

Fig. 2. Bug report template.

The study in [10] analyzed the total time consumed for bug-fixing open-source software by calculating the bug-fixing time for simple bugs that can be resolved in one line of code. In addition, it examined the relationship between the time required to fix the bug and the day of the week the bug report was issued. The traversal method was used to find the defect that was introduced initially or first into the projects. In [11], Ant Colony Optimization (ACO) was applied for bug-triaging feature selection. The major contribution of this study was to optimize the bug assignment step in bug triaging compared to other approaches. The experiments were carried out on Mozilla, Eclipse, and JBoss. In [12], the DENATURE approach was introduced to identify duplicate bugs in a crash, aiming to reduce the bug triaging time and improve software quality.

In [13], a bug triaging mechanism was examined in a distributed environment, where several developers and testers work together remotely from several locations. Using topic modeling and fuzzy sets, the relationship between developers and bugs was developed, and the results were compared with existing machine-learning models. In [14], a self-bug triaging approach using reinforcement learning was proposed. This approach acts like a recommendation system to assign the bug to the appropriate developers. Open-source datasets were used and a comparative analysis was performed with state-of-the-art approaches, which the reinforcement learning approach outperformed. In [15], an approach was developed using the text encoding technique to precisely assign bugs to developers. It used open-source software such as Mozilla and Eclipse and a comparative analysis was performed.

In [16], a raking framework approach was proposed, in which the bug reassignment history was studied and the bug was assigned to the appropriate developer with comparatively fewer tossing events. This method also analyzed the similarity among the textual properties of bug reports. In [17], a bug-triaging method was introduced that correlated the relationship between different bugs. This method implemented Natural Language Processing (NLP) and focused on bugs that had not been resolved for a long time. It used LibreOffice and Mozilla datasets and tried to decrease the bug-fixing time to half of the initial period, giving main priority to blocked bugs. In [18], an effective approach to the deep triage strategy was proposed, which speeded up the training process by adding a dense layer to perform classification. The implementation was done with Gated Recurrent Units (GRUs). This method combined different datasets to perform transfer learning, showing promising results.

In [19], a mechanism for the detection of security-related bugs was proposed, as ignoring these bugs can lead to security breaches or sensitive data exposures. In [20], a collaboration framework was combined with random forest, achieving promising results in bug assignment. This research could be expanded by considering the interactions between the developer and the tester to make the bug-triaging process more accurate. In [21], a bug-triaging method was proposed, in which a token value was generated for each bug based on some parameters. Then, these token values were used to sort and prioritize the bugs. The study in [22] experimented with the concept that the efficiency of bug resolution can be increased if the developer is free to choose the bug related to his expertise. This approach helped to differentiate between active and inactive developers. Additionally, this study proposed using a blockchain-decentralized framework to reduce bug-fixing time. In [23], a method was proposed to determine the appropriate amount of bugs that developers can fix before the next release date of the project considering the time taken to fix the bugs. This method can be called a release-aware bug triaging method.

In [24], a two-step deep neural network algorithm was proposed, comparing the accuracy of bug assignments to the team and the developer. This study implemented a multi-label classification approach from the previous history trend of the bug. In [25], automated bug triaging was performed across nine Ericsson products. A novel approach was developed that triaged the bugs with high confidence. In addition, an analysis of log reports was performed to clear crashes. This approach assigned bugs to the developer team and not to individual developers. Unlike other approaches, this study worked on realistic datasets. The main focus in [26] was to resolve the problems of the Bag Of Words (BOW) model, which does not focus on bug semantics. This study provided a new Deep Bidirectional Recurrent Neural Network with Attention

(DBRNA) approach, which tried to improve accuracy by considering the title as well as the description, unlike the previous models that focused only on the title of the bug report. In [27], an algorithm for automatic bug triaging was presented using topic modeling as an extension of LDA. The various components affected by the bug were also considered. This bug triaging method was followed to list the most eligible developer that can fix the bug. Table I provides details on recent contributions to bug triaging.

TABLE I. RECENT STUDIES ON BUG TRIAGING.

Ref.	Contribution	Outcomes measured
[28]	The proposed model used a heterogenous graph representation technique that had better performance compared to other word-word weighting methods.	Top k accuracy metric
[29]	A novel contextual mutation operator enables real-time or live tracking of bugs in the code.	Quality of test suites and the development of bugs that are realistic and closely resemble errors made by developers.
[30]	An ML algorithm was implemented using the reference data from the previously tested versions. The output was used to analyze the functional as well as non-functional values of the next versions.	Accuracy
[31]	Techniques for feature modification can boost confidence while developing bug prediction models and considerably increase prediction accuracy.	Recall values
[32]	A novel adaptive bug localization algorithm was evaluated on publicly available datasets.	Accuracy and required computational resources
[33]	A model was proposed to predict bug referrals to the designer	Bug triage deep learning convolution
[34]	A tool for bug triage automation was proposed.	Severity-prediction
[35]	A detailed study on Android mobile app bugs reproduction.	Information for reproducing the bugs.
[14]	Online bug assignment to the developer based on the features selected. It calculated the inefficient developers based on the bug resolution count, and the bugs were assigned to efficient developers. It used monthly statistics of active developers.	Top k accuracy metric
[36]	Automated the task of bug reassignment or bug tossing using a multilevel approach, in which the CNN model output was fed to a short-term memory network.	Accuracy and F measure
[37]	Two of the most well-known ML frameworks were used in a fault load benchmark that comprised 113 defects reported by ML developers using GitHub and Stack Overflow	Reproducibility and verifiability of the bugs in machine learning-based systems

The main contributions of this study are the following:

- Proposes a hybrid approach of Embedchain and LLM that leverages the vector embeddings determined by the Embedchain model to further classify the bug report.
- The model was also used to classify the priority of bug reports as high, low, and medium.

II. PROPOSED FRAMEWORK

To improve the efficiency of classifying bugs, a technical approach was proposed that combined Embedchain and an additional LLM model. This framework simplifies the task of managing and testing bug reports by incorporating sophisticated machine-learning techniques. By combining the Embedchain model of data manipulation storage with the expertise of the LLM model in logic and query processing, this approach automates and accelerates the error classification process with faster error reporting processes and better accuracy in classification and testing results, leading to improved error handling. Figure 3 presents the proposed algorithm and describes the flow of the data, i.e., the bug reports to the Embedchain-LLM model.

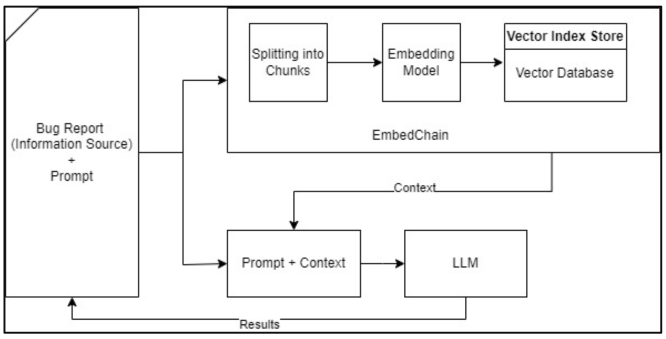


Fig. 3. Proposed Embedchain LLM model for automatic bug triaging.

The proposed method automatically classifies bugs using the Embedchain and LLM models. The steps to perform bug triaging are as follows:

1. Import the data from the Eclipse project sourced from Kaggle.
2. Imported data are input into the Embedchain-LLM model.
3. A specific prompt/query is provided and actions are taken based on this query.
4. Data are processed by the Embedchain and LLM models.

The imported bug report is cleaned, eliminating irrelevant information. The Embedchain model employs a neural network to identify patterns and establish relationships among bugs in imported reports. Each report is divided into smaller segments. The segments are then connected to form a chain and transformed into vectors that are stored in a vector database. The vectors are indexed using the vector indices. The resulting set of vectors represents the data in a numerical format. The prompt/query from Step 3, processed along with the previously converted vector set, is sent to the LLM model. The proposed approach can be summarized as the conversion of bug reports into vector embeddings. Based on the prompt given by the tester, the LLM uses the output of the Embedchain model to perform bug-triaging tasks such as classification, predicting the priority, etc.

III. EXPERIMENTAL RESULTS

The dataset for the Eclipse project was obtained from [38]. Approximately 1,000 reports were used. The bug category was determined using the previously described model. This further reduces the manual effort needed to identify the category and classification of the bugs. The model was further leveraged to find the bug's priority. Each bug report was scanned using the embedding model 'text-embedding-ada-002', which outperformed other models in identifying, comprehending, and classifying defects. Additionally, the ChromaDB vector database was used to store vectors, chosen for its scalability, flexibility, and performance. This ensures that even if the bug report contains extensive data, it can automatically process the entire report to achieve the desired outcomes, eliminating the need to divide the report into smaller segments. Moreover, the Azure Open AI GPT-3.5 Turbo model was employed as LLM to generate the results. Table II shows some classification examples figured out by the model.

TABLE II. CLASSIFICATION OF BUG REPORTS

Group 1: User Interface Defects
- Defect ID: 550153, 550158
Group 2: Performance Defects
- Defect ID: 550156
Group 3: Server Adapter Defects
- Defect ID: 550159
Group 4: Code Completion Defects
- Defect ID: 550160
Group 5: Debugging Defects
- Defect ID: 550165
Group 6: Language Support Defects
- Defect ID: 550166
Group 7: Multi-threading Defects
- Defect ID: 550167
Group 8: Project Navigation Defects
- Defect ID: 550168
Group 9: Build and Compilation Defects
- Defect ID: 550164
Group 10: Dependency Management Defects
- Defect ID: 551129
Group 11: License and Target Update Defects
- Defect ID: 551130
Group 12: Automated Error Reporting Defects
- Defect ID: 551136
Group 13: Memory Management Defects
- Defect ID: 551145
Group 14: Git Setup Defects
- Defect ID: 551141
Group 15: Patch and Update Defects
- Defect ID: 551146
Group 16: User Interface Defects
- Defect ID: 551149

Figure 4 shows the classification distribution of bugs/defects as performed by the Embedchain-LLM model. Figure 5 shows the bug priority distribution using the proposed model, based on the bug description. The model bifurcates bug reports into low, medium, and high priority.

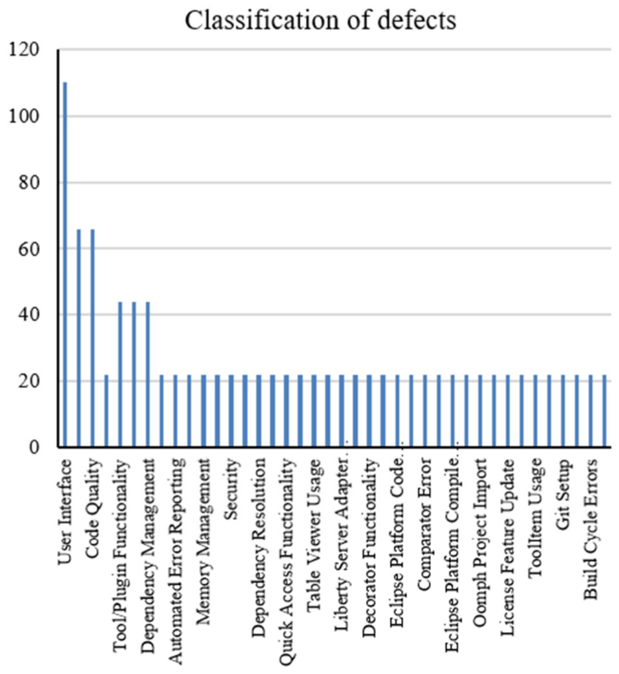


Fig. 4. Classification of bugs using the proposed method.

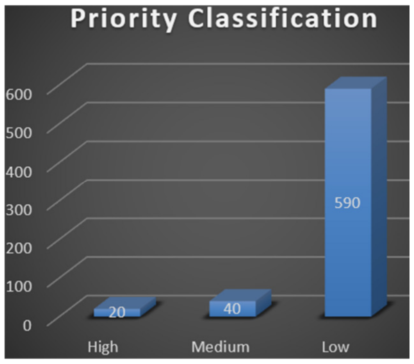


Fig. 5. Priority classification with bug counts.

IV. CONCLUSION

This paper used a hybrid approach based on Embedchain and an LLM for automatic bug triaging. This approach was used to automatically classify bugs and assign priorities. The strengths of both Embedchain and LLM models were used to make the system robust and suitable for real-world applications. Complex high-dimensional data, hierarchical understanding, and enhanced interpretability are the advantages provided by using a hybrid of Embedchain and LLM. This model was also used to classify bug reports and figure out detailed categories. The model provides fault tolerance, making it suitable for noisy and uncertain environments. Second, the model predicts the priority of the bugs based on the context of the bug report, further reducing the cost of manual priority classification. Existing models for bug triaging have certain limitations, such as that they are trained on limited datasets and are not capable of providing on-the-fly results. Additionally, previous techniques that used LLM had fixed token lengths. If the bug report contains more words, the results provided will be inconsistent. The proposed model tries to curb all existing loopholes by providing faster triaging through NLP and vectors to improve accuracy. Furthermore, the model can be enhanced by considering the developer's bug resolution history to assign it to the developer.

Along with these positive results, there are some limitations and challenges, which include complexity in the integration of the model and security vulnerabilities. The LLM used needs to be fine-tuned for accurate prediction. The dataset (bug reports) needs to be well documented for appropriate results. In addition, high computational power is needed to use the proposed hybrid of Embedchain and LLM. In some of the cases, balancing between LLM response and Embedchain embeddings may be inconsistent due to poor vector embeddings. An increase in latency may further degrade the performance of the system, making it less responsive. This approach may face problems in calculating the severity of bug reports. Assessment of severity requires a deep understanding of the bug report context, which can be challenging in some cases. These challenges and limitations must be handled carefully to maximize the strengths and benefits of the model.

REFERENCES

[1] R. Wang, X. Ji, S. Xu, Y. Tian, S. Jiang, and R. Huang, "An empirical assessment of different word embedding and deep learning models for bug assignment," *Journal of Systems and Software*, vol. 210, Apr. 2024, Art. no. 111961, https://doi.org/10.1016/j.jss.2024.111961.

[2] G. Catolino, F. Palomba, A. Zaidman, and F. Ferrucci, "Not all bugs are the same: Understanding, characterizing, and classifying bug types," *Journal of Systems and Software*, vol. 152, pp. 165–181, Jun. 2019, https://doi.org/10.1016/j.jss.2019.03.002.

[3] R. Chen, S.-K. Guo, X. Z. Wang, and T. L. Zhang, "Fusion of Multi-RSMOTE With Fuzzy Integral to Classify Bug Reports With an Imbalanced Distribution," *IEEE Transactions on Fuzzy Systems*, vol. 27, no. 12, pp. 2406–2420, Sep. 2019, https://doi.org/10.1109/TFUZZ.2019.2899809.

[4] G. Parthasarathy, D. C. Tomar, and B. John, "Analysis of Bug Triage using Data Preprocessing (Reduction) Techniques," *International Journal of Computer Applications*, vol. 125, no. 9, pp. 8–15, 2015.

[5] M. N. A. Khan, A. M. Mirza, R. A. Wagan, M. Shahid, and I. Saleem, "A Literature Review on Software Testing Techniques for Smartphone Applications," *Engineering, Technology & Applied Science Research*, vol. 10, no. 6, pp. 6578–6583, Dec. 2020, https://doi.org/10.48084/etasr.3844.

[6] S. Q. Xi, Y. Yao, X. S. Xiao, F. Xu, and J. Lv, "Bug Triaging Based on Tossing Sequence Modeling," *Journal of Computer Science and Technology*, vol. 34, no. 5, pp. 942–956, Sep. 2019, https://doi.org/10.1007/s11390-019-1953-5.

[7] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *Journal of Systems and Software*, vol. 117, pp. 166–184, Jul. 2016, https://doi.org/10.1016/j.jss.2016.02.034.

[8] B. S. Neysiani and S. M. Babamir, "New labeled dataset of interconnected lexical typos for automatic correction in the bug reports," *SN Applied Sciences*, vol. 1, no. 11, Oct. 2019, Art. no. 1385, https://doi.org/10.1007/s42452-019-1419-y.

[9] D. Dreyton, A. A. Araújo, A. Dantas, R. Saraiva, and J. Souza, "A Multi-objective Approach to Prioritize and Recommend Bugs in Open Source Repositories," in *Search Based Software Engineering*, Raleigh, NC, USA, 2016, pp. 143–158, https://doi.org/10.1007/978-3-319-47106-8_10.

[10] E. Eiroa-Lledo, R. H. Ali, G. Pinto, J. Anderson, and E. Linstead, "Large-Scale Identification and Analysis of Factors Impacting Simple Bug Resolution Times in Open Source Software Repositories," *Applied Sciences*, vol. 13, no. 5, Jan. 2023, Art. no. 3150, https://doi.org/10.3390/app13053150.

[11] A. Kukkar *et al.*, "ProRE: An ACO- based programmer recommendation model to precisely manage software bugs," *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 1, pp. 483–498, Jan. 2023, https://doi.org/10.1016/j.jksuci.2022.12.017.

[12] R. Chauhan, S. Sharma, and A. Goyal, "DENATURE: duplicate detection and type identification in open source bug repositories," *International Journal of System Assurance Engineering and Management*, vol. 14, no. 1, pp. 275–292, Mar. 2023, https://doi.org/10.1007/s13198-023-01855-x.

[13] R. R. Panda and N. K. Nagwani, "Topic modeling and intuitionistic fuzzy set-based approach for efficient software bug triaging," *Knowledge and Information Systems*, vol. 64, no. 11, pp. 3081–3111, Nov. 2022, https://doi.org/10.1007/s10115-022-01735-z.

[14] Y. Liu, X. Qi, J. Zhang, H. Li, X. Ge, and J. Ai, "Automatic Bug Triaging via Deep Reinforcement Learning," *Applied Sciences*, vol. 12, no. 7, Jan. 2022, Art. no. 3565, https://doi.org/10.3390/app12073565.

[15] T. W. W. Aung, Y. Wan, H. Huo, and Y. Sui, "Multi-triage: A multi-task learning framework for bug triage," *Journal of Systems and Software*, vol. 184, Feb. 2022, Art. no. 111133, https://doi.org/10.1016/j.jss.2021.111133.

[16] Y. Su *et al.*, "Reducing Bug Triaging Confusion by Learning from Mistakes with a Bug Tossing Knowledge Graph," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Melbourne, Australia, Nov. 2021, pp. 191–202, https://doi.org/10.1109/ASE51524.2021.9678574.

[17] H. Jahanshahi, K. Chhabra, M. Cevik, and A. Başar, "DABT: A Dependency-aware Bug Triaging Method," in *Evaluation and Assessment in Software Engineering*, Trondheim Norway, Jun. 2021, pp. 221–230, https://doi.org/10.1145/3463274.3463342.

[18] E. Tüzün, A. Çetin, and E. Doğan, "An Automated Bug Triaging Approach using Deep Learning: A Replication Study," *European Journal of Science and Technology*, no. 21, pp. 268–274, Jan. 2021, https://doi.org/10.31590/ejosat.781341.

[19] A. D. Sawadogo, Q. Guimard, T. F. Bissyandé, A. K. Kaboré, J. Klein, and N. Moha, "Early Detection of Security-Relevant Bug Reports using Machine Learning: How Far Are We?" arXiv, Dec. 19, 2021, https://doi.org/10.48550/arXiv.2112.10123.

[20] H. Wu, Y. Ma, Z. Xiang, C. Yang, and K. He, "A spatial–temporal graph neural network framework for automated software bug triaging," *Knowledge-Based Systems*, vol. 241, Apr. 2022, Art. no. 108308, https://doi.org/10.1016/j.knosys.2022.108308.

[21] A. Goyal and N. Sardana, "Feature Ranking and Aggregation for Bug Triaging in Open-Source Issue Tracking Systems," in *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, Noida, India, Jan. 2021, pp. 871–876, https://doi.org/10.1109/Confluence51648.2021.9377053.

[22] C. Gupta and M. M. Freire, "A decentralized blockchain oriented framework for automated bug assignment," *Information and Software Technology*, vol. 134, Jun. 2021, Art. no. 106540, https://doi.org/10.1016/j.infsof.2021.106540.

[23] Y. Kashiwa and M. Ohira, "A Release-Aware Bug Triaging Method Considering Developers' Bug-Fixing Loads," *IEICE Transactions on Information and Systems*, vol. E103.D, no. 2, pp. 348–362, Feb. 2020, https://doi.org/10.1587/transinf.2019EDP7152.

[24] C. A. Choquette-Choo, D. Sheldon, J. Proppe, J. Alphonso-Gibbs, and H. Gupta, "A Multi-label, Dual-Output Deep Neural Network for Automated Bug Triaging," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, Boca Raton, FL, USA, Dec. 2019, pp. 937–944, https://doi.org/10.1109/ICMLA.2019.00161.

[25] A. Sarkar, P. C. Rigby, and B. Bartalos, "Improving Bug Triaging with High Confidence Predictions at Ericsson," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Cleveland, OH, USA, Sep. 2019, pp. 81–91, https://doi.org/10.1109/ICSME.2019.00018.

[26] S. Mani, A. Sankaran, and R. Aralikatte, "DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging," in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, Kolkata, India, Jan. 2019, pp. 171–179, https://doi.org/10.1145/3297001.3297023.

[27] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, "Improving Automated Bug Triaging with Specialized Topic Model," *IEEE Transactions on Software Engineering*, vol. 43, no. 3, pp. 272–297, Mar. 2017, https://doi.org/10.1109/TSE.2016.2576454.

[28] S. F. A. Zaidi, H. Woo, and C.-G. Lee, "A Graph Convolution Network-Based Bug Triage System to Learn Heterogeneous Graph Representation of Bug Reports," *IEEE Access*, vol. 10, pp. 20677–20689, 2022, https://doi.org/10.1109/ACCESS.2022.3153075.

[29] C. Richter and H. Wehrheim, "Learning Realistic Mutations: Bug Creation for Neural Bug Detectors," in *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*, Valencia, Spain, Apr. 2022, pp. 162–173, https://doi.org/10.1109/ICST53961.2022.00027.

[30] A. Gartziandia *et al.*, "Machine learning-based test oracles for performance testing of cyber-physical systems: An industrial case study on elevators dispatching algorithms," *Journal of Software: Evolution and Process*, vol. 34, no. 11, 2022, Art. no. e2465, https://doi.org/10.1002/smr.2465.

[31] S. T. Cynthia, B. Roy, and D. Mondal, "Feature Transformation for Improved Software Bug Detection Models," in *15th Innovations in Software Engineering Conference*, Gandhinagar, India, Feb. 2022, pp. 1–10, https://doi.org/10.1145/3511430.3511444.

[32] M. Fejzer, J. Narębski, P. Przymus, and K. Stencel, "Tracking Buggy Files: New Efficient Adaptive Bug Localization Algorithm," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2557–2569, Jul. 2022, https://doi.org/10.1109/TSE.2021.3064447.

[33] R. Sepahvand, R. Akbari, B. Jamasb, S. Hashemi, and O. Boushehrian, "Using word embedding and convolution neural network for bug triaging by considering design flaws," *Science of Computer Programming*, vol. 228, Jun. 2023, Art. no. 102945, https://doi.org/10.1016/j.scico.2023.102945.

[34] O. Picus and C. Serban, "Bugsby: a tool support for bug triage automation," in *Proceedings of the 2nd ACM International Workshop on AI and Software Testing/Analysis*, Jul. 2022, pp. 17–20, https://doi.org/10.1145/3536168.3543301.

[35] J. Johnson, J. Mahmud, T. Wendland, K. Moran, J. Rubin, and M. Fazzini, "An Empirical Investigation into the Reproduction of Bug Reports for Android Apps," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Honolulu, HI, USA, Mar. 2022, pp. 321–322, https://doi.org/10.1109/SANER53432.2022.00048.

[36] J. Jang and G. Yang, "A Bug Triage Technique Using Developer-Based Feature Selection and CNN-LSTM Algorithm," *Applied Sciences*, vol. 12, no. 18, Jan. 2022, Art. no. 9358, https://doi.org/10.3390/app12189358.

[37] M. M. Morovati, A. Nikanjam, F. Tambon, F. Khomh, and Z. M. (Jack) Jiang, "Bug characterization in machine learning-based systems," *Empirical Software Engineering*, vol. 29, no. 1, Dec. 2023, Art. no. 14, https://doi.org/10.1007/s10664-023-10400-0.

[38] S. Kumara, "Software bugs reports." [Online]. Available: https://www.kaggle.com/datasets/samanthakumara/software-bug-reports.

AUTHORS PROFILE

Deepshikha Chhabra is currently working as an Assistant Professor at Chandigarh University, Mohali, Punjab, India. She has more than 11 years of experience, including IT and teaching experience. Her expertise extends to areas such as Software Engineering, Machine Learning, and AI. She has also served at Accenture as a software developer and worked in the business intelligence domain.

Raman Chadha is a seasoned expert in Computer Science & Engineering, boasting over 26 years of academic and professional experience. Currently a Professor at Chandigarh University, Punjab, India, he is deeply involved in advancing his field through active participation in professional bodies such as the Computer Society of India and the International Association of Computer Science and Information Technology. As the Chief Editor of the Journal of Engineering Design and Analysis, he has contributed significantly to the academic literature, particularly in emerging technological domains. He has edited and authored numerous books and holds around 40 patents across computer science, electronics, and mechanical engineering, reflecting his innovative contributions. With more than 80 research papers published in reputable journals indexed in Scopus, Web of Science, and UGC Care, Dr. Chadha's work has garnered recognition, including the Outstanding Researcher Award at NITTTR Chandigarh for his commitment to sustainable development. His expertise is also sought after in hackathons and coding competitions, where he frequently serves as a judge, mentoring aspiring engineers and developers. Dr. Chadha's dedication to continuous learning is evident through his professional certifications in areas like Python, blockchain, full-stack web development, AI, and data science, obtained from platforms such as Coursera and LinkedIn. These credentials underline his commitment to staying at the forefront of technological advancements. Dr. Chadha's illustrious career highlights his contributions to both academia and industry, making him a respected leader in the field of Computer Science & Engineering.