iJRASET
International Journal For Research in Applied Science and Engineering Technology

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 14 **Issue:** VII **Month of publication:** July 2026

**DOI:** https://doi.org/10.22214/ijraset.2026.84447

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 14 Issue VII July 2026- Available at www.ijraset.com*

# Intelligent Bug Triaging Using CodeBERT, Graph Attention Networks, FAISS, and Explainable Machine Learning

Pogiri Pujitha[1], Gangadhar Doma[2]
[1]*Student, Master of Computer Applications (MCA), AQJ Centre for P.G. Studies (Affiliated to Andhra University, Visakhapatnam), Visakhapatnam, Andhra Pradesh, India*
[2]*Associate Professor, Department of MCA, AQJ Centre for P.G. Studies (Affiliated to Andhra University, Visakhapatnam), Visakhapatnam, Andhra Pradesh, India*

***Abstract:*** ***Software bug triaging is a crucial software maintenance activity that involves assigning reported bugs to appropriate developers and determining their priority. Manual bug triaging is often time-consuming, error-prone, and inefficient for large-scale software projects due to the increasing volume of bug reports. This paper presents an intelligent bug triaging framework that integrates CodeBERT, Graph Attention Networks (GAT), FAISS, XGBoost, and SHAP to automate bug analysis and improve decision-making. Initially, bug reports are preprocessed and transformed into contextual semantic embeddings using CodeBERT. FAISS performs efficient similarity search to retrieve related historical bug reports, while GAT captures relationships among bug reports, developers, and software components to enhance developer recommendation. The extracted semantic and graph-based features are combined and supplied to an XGBoost classifier for bug priority prediction. To improve model transparency, SHAP is employed to explain the contribution of individual features to each prediction. The proposed framework is deployed through a Streamlit-based web application that provides an interactive interface for bug analysis and recommendation. By combining semantic understanding, graph learning, similarity retrieval, and explainable artificial intelligence, the proposed system improves the efficiency, accuracy, and interpretability of automated bug triaging, making it suitable for large-scale software maintenance environments.***

***Keywords— Bug Triaging, Software Maintenance, CodeBERT, Graph Attention Network, FAISS, XGBoost, SHAP, Explainable Artificial Intelligence, Machine Learning, Streamlit.***

## I. INTRODUCTION

Software maintenance is one of the most important phases of the Software Development Life Cycle (SDLC), accounting for a significant portion of the overall development effort. A major activity during software maintenance is bug triaging, which involves analyzing reported software defects, identifying their severity and priority, and assigning them to the most suitable developers for resolution. Efficient bug triaging is essential for reducing bug resolution time, improving software quality, and ensuring the timely delivery of software products.

With the rapid growth of large-scale software projects and open-source repositories, thousands of bug reports are submitted daily through issue tracking systems such as Bugzilla, Jira, and GitHub Issues. Manual bug triaging is labor-intensive, time-consuming, and often inconsistent, as project managers must analyze each report and identify the developer with the appropriate expertise. As the number and complexity of bug reports increase, manual assignment becomes increasingly difficult, leading to delayed bug resolution and higher maintenance costs. Several automated bug triaging techniques have been proposed using traditional machine learning algorithms, including Naïve Bayes, Support Vector Machines (SVM), Random Forests, and Decision Trees. These approaches primarily rely on handcrafted textual features such as Bag-of-Words (BoW) and Term Frequency–Inverse Document Frequency (TF-IDF). Although they reduce manual effort, they often fail to capture the contextual meaning of bug reports and struggle with complex software engineering terminology, resulting in reduced prediction accuracy.

Recent advances in transformer-based language models have significantly improved semantic understanding in software engineering tasks. CodeBERT, a pre-trained model trained on both programming language and natural language corpora, generates contextual embeddings that effectively represent software bug reports. These embeddings preserve semantic information and enable better identification of similar bugs compared with conventional text representation methods.

Besides textual semantics, the relationships among developers, software components, and historical bug reports play a vital role in developer recommendation. Graph Attention Networks (GATs) are capable of learning these structural relationships by assigning adaptive attention weights to neighboring nodes, thereby capturing developer expertise and collaboration patterns more effectively than traditional graph-based approaches.

To further improve the efficiency of bug triaging, the proposed framework integrates Facebook AI Similarity Search (FAISS) for fast retrieval of semantically similar historical bug reports. This enables efficient searching in large software repositories and assists in identifying previously resolved issues that are relevant to newly reported bugs. In addition, an XGBoost classifier is employed to predict bug priority using semantic and graph-based features. To enhance transparency and user trust, SHAP (SHapley Additive Explanations) is incorporated to explain the contribution of individual features to each prediction.

Based on these technologies, this paper proposes an Intelligent Bug Triaging Framework that combines CodeBERT, FAISS, Graph Attention Networks, XGBoost, and SHAP into a unified architecture. The framework automates semantic feature extraction, similar bug retrieval, developer recommendation, priority prediction, and explainable decision support. A Streamlit-based web application is developed to provide an interactive interface for practical software maintenance.

The major contributions of this work are summarized as follows:

- Development of an automated bug triaging framework using CodeBERT for contextual semantic feature extraction.
- Integration of FAISS for efficient semantic retrieval of historical bug reports.
- Utilization of Graph Attention Networks to model relationships among developers, bug reports, and software components.
- Implementation of an XGBoost classifier for accurate bug priority prediction.
- Incorporation of SHAP to provide interpretable explanations for prediction results.
- Deployment of the complete framework through a Streamlit-based web application for real-time bug analysis and recommendation.

The remainder of this paper is organized as follows. Section II reviews existing research on automated bug triaging. Section III describes the proposed methodology. Section IV presents the system architecture and algorithms. Section V discusses the experimental setup and results. Finally, Section VI concludes the paper and outlines future research directions.

## II. RELATED WORK

Automated bug triaging has been widely studied to reduce the manual effort involved in assigning software bugs to appropriate developers. Early approaches treated bug triaging as a text classification problem using traditional machine learning algorithms such as Naïve Bayes, Support Vector Machines (SVM), Decision Trees, and Random Forests. These methods commonly employed feature extraction techniques such as Bag-of-Words (BoW) and Term Frequency–Inverse Document Frequency (TF-IDF). Although they improved the efficiency of bug assignment, they were limited in capturing the semantic context of bug reports and often produced lower accuracy for large and complex software repositories.

The emergence of deep learning introduced more effective techniques for bug report analysis. Models based on Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks automatically learned textual features from bug descriptions and demonstrated better performance than conventional machine learning approaches. However, these models primarily focused on textual information and did not fully exploit the relationships among developers, software components, and historical bug reports.

Recent advancements in transformer-based language models have significantly enhanced software engineering applications. CodeBERT, pre-trained on both source code and natural language, generates contextual embeddings that effectively represent software bug reports. Compared with traditional feature extraction methods, CodeBERT captures semantic relationships more accurately, leading to improved bug classification, code search, and developer recommendation performance.

Graph-based learning techniques have also gained attention in automated bug triaging. Graph Attention Networks (GATs) model the relationships among bug reports, developers, and software components by assigning adaptive attention weights to neighboring nodes. This enables the framework to learn developer expertise and collaboration patterns more effectively than conventional graph-based methods, thereby improving recommendation accuracy.

Efficient retrieval of similar historical bug reports is another important aspect of intelligent bug triaging. Facebook AI Similarity Search (FAISS) provides scalable vector indexing and approximate nearest-neighbor search for high-dimensional embeddings. By retrieving semantically similar historical bugs, FAISS assists in identifying relevant developer assignments and accelerates the bug triaging process.

Despite these advances, most existing studies focus on only one or two aspects of automated bug triaging, such as semantic feature extraction, graph learning, or priority prediction. Very few approaches integrate transformer-based embeddings, graph neural networks, semantic similarity search, explainable artificial intelligence, and interactive deployment into a single framework.

To address these limitations, this paper proposes an integrated bug triaging framework that combines CodeBERT for semantic feature extraction, FAISS for similarity retrieval, Graph Attention Networks for developer recommendation, XGBoost for bug priority prediction, and SHAP for explainable decision-making. The complete framework is deployed through a Streamlit application, providing an efficient and interpretable solution for intelligent software bug triaging.

## III. PROPOSED METHODOLOGY

The proposed Intelligent Bug Triaging Framework integrates transformer-based semantic learning, graph neural networks, semantic similarity search, machine learning, and explainable artificial intelligence to automate software bug analysis and developer recommendation. The overall workflow consists of six major stages: data preprocessing, semantic feature extraction, similarity retrieval, graph-based learning, bug priority prediction, and explainable decision support. Figure 1 illustrates the overall architecture of the proposed system.

### *A. Data Preprocessing*

Bug reports collected from software repositories contain information such as bug summary, description, product, component, severity, priority, and assigned developer. Before model training, the reports are preprocessed to improve data quality. The preprocessing steps include duplicate removal, handling missing values, text normalization, tokenization, stop-word removal, and conversion of text into a standardized format. These operations reduce noise and prepare the data for semantic feature extraction.

### *B. Semantic Feature Extraction using CodeBERT*

The preprocessed bug reports are transformed into contextual semantic embeddings using CodeBERT, a transformer-based language model pre-trained on both programming language and natural language corpora. Unlike traditional feature extraction methods such as TF-IDF, CodeBERT captures the contextual meaning of software-related text and produces dense vector representations.

For a given bug report B, the semantic embedding is represented as:

[
E = \text{CodeBERT}(B)
]

where E denotes the contextual embedding generated from the bug report. These embeddings provide a rich semantic representation that improves similarity analysis and downstream prediction tasks.

### *C. Similar Bug Retrieval using FAISS*

To identify previously resolved issues, the generated embeddings are indexed using Facebook AI Similarity Search (FAISS). FAISS performs efficient Approximate Nearest Neighbor (ANN) search and retrieves the top-*k* semantically similar historical bug reports. This process helps identify duplicate or related issues and provides valuable historical information for developer recommendation.

### *D. Graph Construction and Graph Attention Network*

Software maintenance involves complex relationships among bug reports, developers, and software components. To model these interactions, a heterogeneous graph is constructed in which nodes represent bug reports, developers, and software modules, while edges represent assignment and dependency relationships. The constructed graph is processed using a Graph Attention Network (GAT). The attention mechanism assigns different importance weights to neighboring nodes, allowing the model to learn developer expertise and collaboration patterns more effectively. The graph representations are combined with semantic embeddings to generate a comprehensive feature representation for each bug report.

### *E. Bug Priority Prediction using XGBoost*

The semantic features obtained from CodeBERT and the graph representations learned by the GAT are fused into a single feature vector. This integrated representation is provided as input to an XGBoost classifier, which predicts the priority level of each bug report. XGBoost is selected because of its high prediction accuracy, robustness, scalability, and ability to model complex nonlinear relationships. The predicted priority assists project managers in scheduling bug resolution based on urgency.

### F. Explainable Prediction using SHAP

Although machine learning models achieve high predictive performance, their decisions are often difficult to interpret. To improve transparency, the proposed framework incorporates SHAP (SHapley Additive Explanations). SHAP quantifies the contribution of each feature to the final prediction and identifies the factors that most influence developer recommendation and priority prediction. This improves user trust and supports informed decision-making during software maintenance.

### G. Streamlit-Based Deployment

The complete framework is deployed using a Streamlit web application. The interface enables users to upload or enter bug reports, retrieve semantically similar historical issues, obtain developer recommendations, predict bug priorities, and visualize SHAP-based explanations. This interactive deployment demonstrates the practical applicability of the proposed framework for real-world software development environments.

### H. Workflow of the Proposed Framework

The overall workflow of the proposed methodology is summarized as follows:

1) Collect and preprocess software bug reports.
2) Generate semantic embeddings using CodeBERT.
3) Retrieve similar historical bugs using FAISS.
4) Construct a developer–bug interaction graph.
5) Learn graph representations using the Graph Attention Network.
6) Fuse semantic and graph-based features.
7) Predict bug priority using XGBoost.
8) Generate SHAP explanations for prediction results.
9) Display recommendations through the Streamlit application.

The proposed methodology combines semantic understanding, graph-based learning, efficient similarity retrieval, and explainable machine learning into a unified framework. This integrated approach enhances developer recommendation accuracy, improves bug priority prediction, reduces manual triaging effort, and provides transparent decision support for modern software maintenance systems.

## IV. SYSTEM ARCHITECTURE AND ALGORITHMS

### A. System Architecture

The proposed Intelligent Bug Triaging Framework integrates semantic feature extraction, graph learning, similarity retrieval, machine learning, and explainable artificial intelligence into a unified architecture. The system is designed to automate developer recommendation and bug priority prediction while providing interpretable results. The overall architecture is shown in Figure 1.

Initially, software bug reports are collected from issue tracking repositories and passed through a preprocessing module that removes duplicate records, handles missing values, and normalizes textual data. The cleaned bug reports are then processed using CodeBERT, which generates contextual semantic embeddings representing the meaning of each bug report.

The generated embeddings are indexed using FAISS, enabling efficient retrieval of semantically similar historical bug reports. Simultaneously, a heterogeneous graph is constructed to represent relationships among bug reports, developers, and software components. This graph is processed using a Graph Attention Network (GAT), which learns developer expertise by assigning adaptive attention weights to neighboring nodes.

The semantic embeddings and graph-based features are combined into a unified feature representation and provided to an XGBoost classifier for bug priority prediction. Finally, SHAP generates feature importance scores to explain the prediction results. The complete framework is deployed through a Streamlit web application that provides an interactive interface for bug analysis, developer recommendation, and priority prediction.

### B. System Modules

The proposed architecture consists of the following major modules:

1) Data Preprocessing: Cleans and normalizes bug reports by removing duplicate records, handling missing values, and preparing textual data for semantic analysis.

2) CodeBERT Feature Extraction: Converts each bug report into a contextual embedding that captures semantic information from software-related text.
3) FAISS Similarity Retrieval: Retrieves semantically similar historical bug reports using Approximate Nearest Neighbor (ANN) search to support developer recommendation.
4) Graph Attention Network: Models relationships among developers, bug reports, and software components to learn developer expertise and collaboration patterns.
5) XGBoost Prediction: Predicts the priority level of each bug report using the combined semantic and graph-based features.
6) SHAP Explainability: Explains the contribution of individual features to each prediction, improving transparency and user trust.
7) Streamlit Deployment: Provides a user-friendly interface for uploading bug reports, viewing recommendations, and visualizing prediction results.

### C. *Proposed Algorithm*

The overall bug triaging process is summarized in Algorithm 1.

Algorithm 1: Intelligent Bug Triaging Framework

Input: Bug Report B

Output: Recommended Developer, Bug Priority, SHAP Explanation

1) Load historical bug dataset.
2) Preprocess the input bug report.
3) Generate semantic embedding using CodeBERT.
4) Retrieve similar historical bug reports using FAISS.
5) Construct the developer–bug interaction graph.
6) Learn graph representations using the Graph Attention Network.
7) Fuse semantic and graph features.
8) Predict bug priority using XGBoost.
9) Generate SHAP explanations for the prediction.
10) Display developer recommendation and priority through the Streamlit interface.

### D. *Workflow of the Proposed System*

The proposed framework performs bug triaging through the following sequence:

1) Collect software bug reports.
2) Preprocess textual information.
3) Generate CodeBERT embeddings.
4) Retrieve similar bugs using FAISS.
5) Learn developer relationships using GAT.
6) Combine semantic and graph-based features.
7) Predict bug priority using XGBoost.
8) Explain prediction results using SHAP.
9) Display recommendations through the Streamlit application.

### E. *Advantages of the Proposed Architecture*

The proposed architecture offers several advantages over conventional bug triaging approaches:

1) Captures contextual semantics using CodeBERT.
2) Retrieves similar historical bugs efficiently using FAISS.
3) Learns developer expertise through Graph Attention Networks.
4) Predicts bug priority with high accuracy using XGBoost.
5) Improves model transparency using SHAP explanations.
6) Provides an interactive and scalable deployment through Streamlit.
7) Reduces manual effort and supports faster software maintenance.

The integration of semantic language models, graph neural networks, similarity search, explainable machine learning, and interactive deployment provides a comprehensive and efficient solution for intelligent software bug triaging.

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 14 Issue VII July 2026- Available at www.ijraset.com*

## V. EXPERIMENTAL SETUP AND RESULTS

### *A. Experimental Setup*

The proposed Intelligent Bug Triaging Framework was implemented to evaluate its effectiveness in automated developer recommendation and bug priority prediction. The implementation integrates CodeBERT, FAISS, Graph Attention Networks (GAT), XGBoost, and SHAP within a unified software maintenance framework. The experiments were conducted using Python and modern machine learning libraries.

Software Environment

| Component | Technology |
|---|---|
| Programming Language | Python 3.x |
| Deep Learning Framework | PyTorch |
| Transformer Model | CodeBERT |
| Graph Learning | PyTorch Geometric |
| Similarity Search | FAISS |
| Machine Learning | XGBoost |
| Explainable AI | SHAP |
| Web Framework | Streamlit |

Hardware Environment

| Component | Specification |
|---|---|
| Processor | Intel Core i7 or Equivalent |
| RAM | 16 GB |
| Storage | SSD |
| Operating System | Windows/Linux |

The modular architecture enables execution on both CPU and GPU environments.

### *B. Dataset*

The framework was evaluated using historical bug reports collected from publicly available software repositories. Each bug report contains attributes such as bug summary, description, component, priority, severity, assigned developer, and resolution status. Before training, the dataset was preprocessed by removing duplicate records, handling missing values, and normalizing textual information.

The dataset was divided into:

- Training Set: 80%
- Testing Set: 20%

This split ensures reliable evaluation of the proposed framework on unseen bug reports.

### *C. Evaluation Metrics*

The performance of the proposed framework was evaluated using standard classification metrics.

- Accuracy: Measures the percentage of correctly classified bug reports.
- Precision: Indicates the proportion of correctly predicted positive instances.
- Recall: Measures the ability of the model to identify relevant bug reports.
- F1-Score: Represents the harmonic mean of Precision and Recall.
- Retrieval Time: Measures the efficiency of FAISS in retrieving similar bug reports.
- Prediction Time: Evaluates the time required for developer recommendation and bug priority prediction.

These metrics provide a comprehensive assessment of the effectiveness and efficiency of the proposed system.

### D. Experimental Results

The experimental results demonstrate that the proposed framework effectively combines semantic feature extraction, graph learning, similarity retrieval, and explainable machine learning for intelligent bug triaging.

CodeBERT generated meaningful contextual embeddings that improved the semantic representation of bug reports compared with traditional text-based methods. FAISS enabled efficient retrieval of semantically similar historical bugs, reducing the time required to identify previously resolved issues.

The Graph Attention Network (GAT) effectively captured relationships among developers, software components, and historical bug reports, resulting in improved developer recommendation performance. By combining semantic embeddings with graph-based representations, the system produced more accurate recommendations than approaches relying solely on textual features.

The XGBoost classifier accurately predicted bug priority levels using the integrated feature representation. In addition, the SHAP explanation module provided clear insights into the contribution of different features, improving the transparency and interpretability of prediction results.

### E. Comparative Analysis

Table I presents a comparison between the proposed framework and commonly used automated bug triaging approaches.

Table I. Comparison of Bug Triaging Approaches

| Method | Semantic Understanding | Graph Learning | Similarity Retrieval | Explainability | Deployment |
|---|---|---|---|---|---|
| TF-IDF + SVM | Low | No | No | No | No |
| CNN | Moderate | No | No | No | Limited |
| LSTM | Moderate | No | No | No | Limited |
| BERT | High | No | No | Limited | Limited |
| Graph Neural Network | Moderate | Yes | No | No | Limited |
| Proposed Framework | Very High | Yes | Yes (FAISS) | Yes (SHAP) | Yes (Streamlit) |

The proposed framework integrates multiple advanced techniques within a single architecture, enabling improved semantic understanding, efficient retrieval, graph-based learning, and explainable predictions.

### F. Discussion

The experimental study demonstrates that combining transformer-based semantic embeddings with graph neural networks enhances automated bug triaging. CodeBERT effectively captures the contextual meaning of bug reports, while FAISS accelerates similarity retrieval from large repositories. The Graph Attention Network models developer expertise through historical interactions, improving recommendation quality. Furthermore, SHAP increases user confidence by explaining prediction decisions, making the framework more suitable for practical software maintenance environments.

Overall, the proposed framework provides an efficient, scalable, and interpretable solution for automated bug triaging, supporting software teams in reducing manual effort and improving bug resolution efficiency.

## VI. CONCLUSION AND FUTURE WORK

### A. Conclusion

This paper presented an Intelligent Bug Triaging Framework that integrates CodeBERT, FAISS, Graph Attention Networks (GAT), XGBoost, and SHAP to automate software bug analysis and developer recommendation. The proposed framework combines semantic feature extraction, graph-based learning, similarity retrieval, bug priority prediction, and explainable artificial intelligence within a unified architecture.

CodeBERT generates contextual embeddings that effectively capture the semantic meaning of bug reports, while FAISS enables efficient retrieval of similar historical issues from large software repositories. The Graph Attention Network models relationships among developers, bug reports, and software components, improving developer recommendation accuracy. Furthermore, the XGBoost classifier provides reliable bug priority prediction, and SHAP enhances transparency by explaining the contribution of individual features to each prediction. The Streamlit-based web application demonstrates the practical applicability of the proposed framework in supporting intelligent software maintenance.

Overall, the proposed approach reduces manual bug triaging effort, improves recommendation quality, enhances prediction interpretability, and provides an efficient solution for large-scale software development environments.

### *B. Future Work*

Although the proposed framework demonstrates promising performance, several improvements can be explored in future research. The framework can be extended by integrating advanced Large Language Models (LLMs) to further improve semantic understanding of complex bug reports. Dynamic graph neural networks may also be investigated to capture evolving developer expertise and project relationships over time.

Future work can focus on integrating the framework with issue tracking platforms such as Bugzilla, Jira, and GitHub Issues for real-time bug triaging and automated developer assignment. Additional factors, including developer workload, availability, and historical performance, may also be incorporated to improve recommendation quality.

Furthermore, evaluating the framework on larger industrial datasets and exploring federated or privacy-preserving learning techniques would enhance scalability and practical adoption. These improvements have the potential to further increase the efficiency, accuracy, and reliability of intelligent software bug triaging systems.

## VII. ACKNOWLEDGMENT

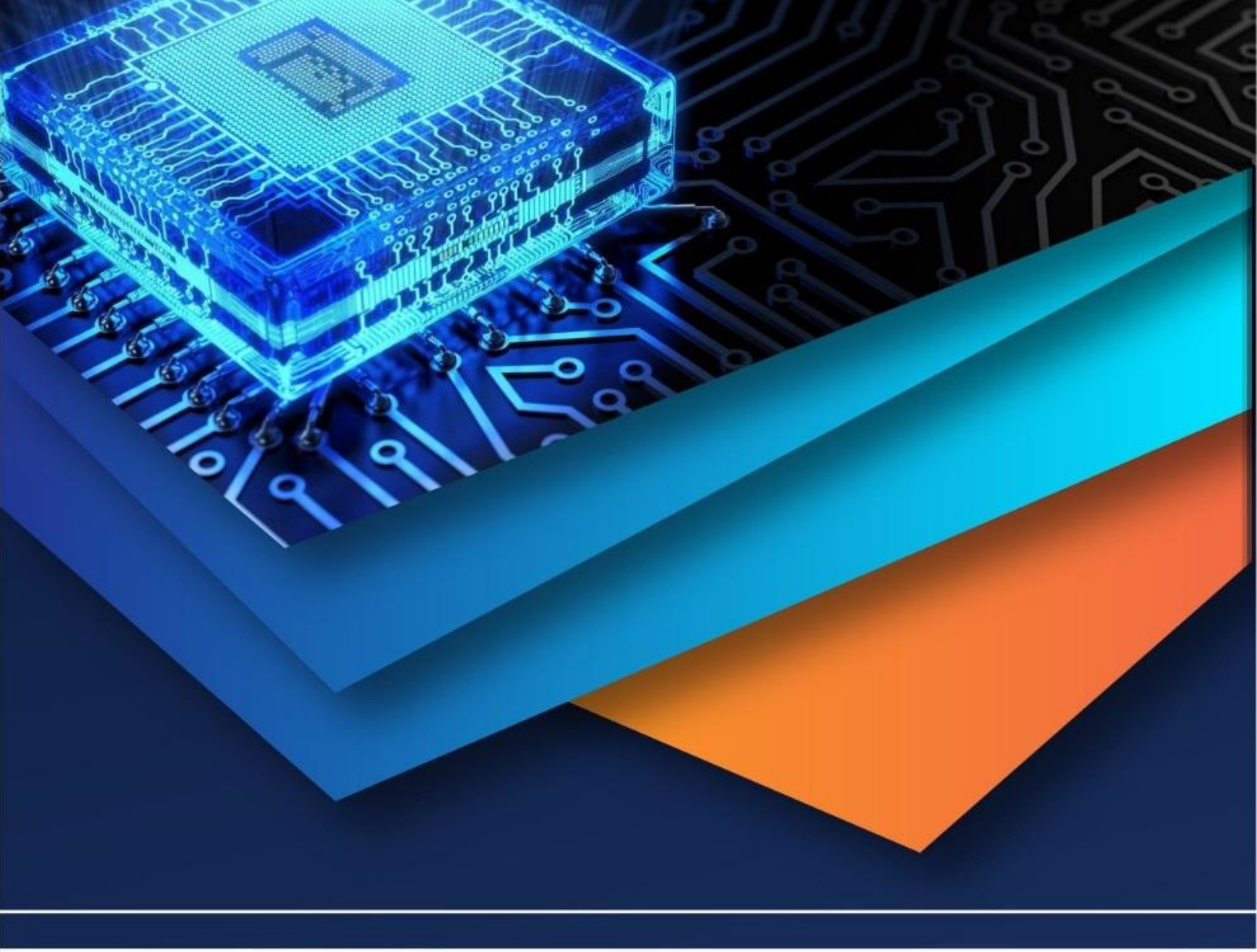
The authors would like to express their sincere gratitude to all researchers and developers whose publicly available datasets and open-source software libraries contributed to this work. Special thanks are extended to the developers of CodeBERT**,** FAISS**,** PyTorch**,** PyTorch Geometric**,** XGBoost**,** SHAP, and Streamlit for providing powerful tools that enabled the implementation of the proposed intelligent bug triaging framework.

The authors also acknowledge the support and encouragement received from their institution, faculty members, and colleagues during the development and evaluation of this research. Their valuable guidance and constructive suggestions significantly contributed to the successful completion of this work.

## REFERENCES

[1] F. Long and M. Rinard, "Automatic Patch Generation by Learning Correct Code," Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2016.
[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," International Conference on Learning Representations (ICLR), 2013.
[3] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Proceedings of NAACL-HLT, 2019.
[4] Z. Feng, D. Guo, D. Tang et al., "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," Proceedings of EMNLP: Findings, 2020.
[5] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph Attention Networks," International Conference on Learning Representations (ICLR), 2018.
[6] J. Johnson, M. Douze, and H. Jégou, "Billion-scale Similarity Search with GPUs," IEEE Transactions on Big Data, vol. 7, no. 3, pp. 535–547, 2021.
[7] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.
[8] S. M. Lundberg and S. Lee, "A Unified Approach to Interpreting Model Predictions," Advances in Neural Information Processing Systems (NeurIPS), 2017.
[9] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.
[10] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," Advances in Neural Information Processing Systems (NeurIPS), 2019.
[11] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," ICLR Workshop on Representation Learning on Graphs, 2019.
[12] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," Proceedings of ACM SIGKDD, 2019.
[13] Streamlit Inc., "Streamlit: The Fastest Way to Build Data Applications," 2024.
[14] Mozilla Foundation, "Bugzilla Issue Tracking System," Available: https://bugzilla.mozilla.org.
[15] Apache Software Foundation, "Apache Hadoop Issue Repository," Available: https://issues.apache.org.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY