ISLAMIC UNIVERSITY OF TECHNOLOGY

Method-Based and BERT-Based Bug Prediction in Open Source Software: A Comparative Study

By

Sadman Mehedi Sivan (190042143)

Mohammad Abir (190042150)

Supervisor Assistant Professor

Nazmul Hossain Dept. of CSE, IUT

A thesis submitted in partial fulfilment of the requirements for the degree of B.Sc. in Software Engineering

Academic Year: 2022-2023

Software Engineering Lab (SEL)

Department of Computer Science and Engineering

Islamic University of Technology (IUT)

A Subsidiary Organ of the Organization of Islamic Cooperation.

Dhaka, Bangladesh.

June 2023

Declaration of Authorship

We, Sadman Mehedi Sivan and Abir Hossain declare that this thesis titled, 'Method-Based and BERT-Based Bug Prediction in Open Source Software: A Comparative Study' and the work presented in it is our own. We confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Any part of this thesis has not been submitted for any other degree or qualification at this University or any other institution.
- Where we have consulted the published work of others, those were clearly attributed.

Submitted By:

(Signature of the Candidate)
Sadman Mehedi Sivan - 190042143
Mohammad Abir - 190042150
July 2024

Method-Based and BERT-Based Bug Prediction in Open Source Software: A Comparative Study

Approved By:

Nazmul Haque
Thesis Supervisor,
Assistant Professor,
Department of Computer Science and Engineering,
Islamic University of Technology (IUT), Dhaka, Bangladesh.

Acknowledgements

I would like to express my heartfelt thanks to Allah Subhanu Wata'ala for giving me the strength to finish this study and being with me when no one else was.

I am grateful to my loved ones for supporting me from the beginning to the end of my research work. I also extend my sincere gratitude to my thesis supervisor Md Nazmul Haque for his constant encouragement and support throughout my study.

I would not have been able to complete this research without the guidance and assistance of several individuals who contributed in various ways. I would like to acknowledge and appreciate their valued help.

Abbreviations

BERT	**B**idirectional **E**ncoder **R**epresentations from **T**ransformers
GPT	**G**enerative **P**re-trained **T**ransformer
SRS	**S**oftware **R**equirement **S**pecification
LSTM	**L**ong **S**hort-**T**erm **M**emory
RNN	**R**ecurrent **N**eural **N**etwork
XGB	**X**treme **G**radient **B**oosting
LightGBM	**Light G**radient **B**oosting **M**achine
SVM	**S**upport **V**ector **M**achine
LLM	**L**arge **L**anguage **M**odel
SRS	**S**oftware **R**equirements **S**pecification
NLP	**N**atural **L**anguage **P**rocessing
RoBERTa	**R**obustly **O**ptimized **BERT A**pproach
DistilBERT	**Distil**led **BERT**

Dedicated to our parents for their continuous encouragement of my academic endeavors and research ...

Abstract

Bug prediction is essential for maintaining software quality, but manual processes are inefficient and error-prone. This study compares traditional method-level bug prediction techniques with modern NLP approaches, focusing on pre-trained language models like BERT. Using a GitHub issues dataset, we analyze the effectiveness, challenges, and practical implications of both methodologies. Our findings show that while method-level approaches remain robust when historical data is available, BERT-based models provide competitive performance and unique advantages, particularly in scenarios lacking detailed code history.

Chapter 1

Introduction

Software bugs are inevitable in large-scale open source projects. As projects grow in size and complexity, identifying and fixing bugs early becomes increasingly critical. Efficient bug prediction not only accelerates the development lifecycle but also reduces overall project costs, minimizes technical debt, and enhances software reliability.

Traditionally, bug prediction techniques have relied on structural and historical data, such as code metrics (e.g., cyclomatic complexity, churn rate, lines of code) and version control logs. These method-level approaches have proven effective, particularly in enterprise environments where such data is consistently maintained.

However, a major limitation of traditional methods is their dependence on accurate and extensive historical data, which may not always be available, especially in rapidly evolving or newly initiated open source projects. Moreover, manual linking between bug reports and code changes is time-consuming and error-prone, reducing the scalability of such approaches.

In recent years, Natural Language Processing (NLP) has introduced a paradigm shift in bug prediction research. The availability of large textual datasets — such as issue titles, descriptions, commit messages, and developer comments — has opened new avenues for automated analysis. Pre-trained language models like BERT (Bidirectional Encoder Representations from Transformers) have demonstrated significant success in capturing the semantic context of text, making them highly suitable for tasks like bug classification, severity estimation, and triage.

This thesis explores and compares two distinct paradigms of bug prediction: the conventional method-level models that rely on handcrafted features, and modern transformer-based NLP models that utilize rich semantic representations. By leveraging real-world datasets and experimental evaluations, this study aims to identify the strengths, weaknesses, and potential synergy between these two approaches in enhancing software quality and maintainability.

Chapter 2

Research Questions

In the context of this research, we aim to investigate the evolving landscape of bug prediction, particularly the comparative performance and practicality of modern NLP-based models versus traditional method-level techniques. With the emergence of powerful pre-trained language models, a core objective of this study is to explore their effectiveness in real-world software engineering scenarios.

The following research questions were formulated to guide our study:

1. **How does a pre-trained language model like BERT perform in bug prediction compared to traditional method-level approaches?**
   This question investigates the capability of transformer-based models, particularly BERT, in accurately classifying and predicting bug-related issues using textual data such as issue titles, descriptions, and commit messages. It also involves a direct performance comparison with classical models that utilize hand-engineered features from code metrics and version control logs.

2. **What are the strengths and limitations of NLP-based versus method-level bug prediction?**
   This question focuses on understanding the practical trade-offs between the two paradigms. While NLP models may offer higher accuracy in semantically rich environments, they often require more computational resources. On the other hand, method-level approaches are more interpretable and lightweight but may struggle in the absence of well-structured code history or labeled datasets. This question explores these contrasts in terms of data dependency, generalizability, resource efficiency, and interpretability.

The answers to these questions will provide insight into when and how to apply each approach, offering recommendations for practitioners and researchers working in automated software quality assurance.

Chapter 3

Motivation

There is a noticeable gap in comprehensive, empirical comparisons between traditional method-level bug prediction techniques and modern NLP-based approaches. While both paradigms have independently shown promising results in software engineering research, few studies offer a side-by-side evaluation under controlled conditions using the same dataset and evaluation criteria.

Traditional method-level models rely on metrics such as code complexity, code churn, developer activity, and change history. These models are generally effective when extensive project metadata and version control information are available. However, their performance tends to degrade in early-stage projects or decentralized open-source environments where such structured data may be sparse or inconsistent.

On the other hand, Natural Language Processing (NLP) models — particularly those based on transformer architectures like BERT — offer a data-driven alternative by leveraging the textual information embedded in issue reports, commit messages, and documentation. These models can potentially overcome the limitations of traditional methods by understanding the context and semantics of natural language descriptions associated with bugs.

Despite their potential, NLP models have not yet been widely adopted in real-world bug prediction pipelines, partly due to concerns over computational overhead, lack of interpretability, and limited comparative evaluation. This study is motivated by the need to bridge this gap and empirically evaluate how these two approaches perform under similar conditions.

By conducting a detailed comparison, we aim to uncover insights that can guide researchers and practitioners in selecting the appropriate bug prediction technique based on their project environment, data availability, and computational constraints.

Chapter 4

Problem Statement

Software systems are becoming increasingly complex, with open source platforms witnessing rapid development cycles, distributed teams, and high user engagement. In this dynamic environment, the ability to predict bugs before they manifest in production code is crucial for maintaining software reliability, reducing maintenance costs, and preserving developer productivity.

- **How can we develop a bug prediction solution that reduces non-conformance costs?**
  Non-conformance costs refer to the financial and resource-related penalties incurred when software fails to meet expected quality standards. These costs can arise from late-stage debugging, production failures, user dissatisfaction, or delays in deployment. Developing an effective bug prediction model that detects potential faults early in the development process can significantly mitigate these costs. The challenge lies in designing models that are accurate, generalizable, and efficient — especially in resource-constrained or data-scarce environments.

- **What are the limitations of traditional solutions, and how can modern approaches address these drawbacks?**
  Traditional bug prediction models typically rely on manually extracted code metrics and historical data. While these models are interpretable and lightweight, they depend heavily on the availability and consistency of structured data — which may not always be feasible, especially in early-phase or open source projects. Moreover, these models often fail to capture the semantic meaning embedded in textual artifacts like issue reports or developer comments.

In contrast, modern NLP-based approaches such as BERT leverage contextual embeddings from natural language inputs. These models offer a more flexible, text-driven framework that can learn patterns from the way developers describe bugs, features, and discussions. However, their adoption remains limited due to concerns around computational demand and interpretability.

This thesis seeks to analyze these trade-offs and propose a practical bug prediction framework that harnesses the strengths of modern NLP techniques while acknowledging the situational advantages of traditional models.

Chapter 5

Research Objectives

The primary objective of this research is to explore and evaluate the potential of advanced Natural Language Processing (NLP) models for the task of software bug prediction. By doing so, we aim to bridge the gap between traditional metric-based techniques and emerging language-driven methods. The following are the core objectives of this study:

- **Implement a BERT-based bug prediction model using NLP.**
  This objective focuses on the design, fine-tuning, and implementation of a bug prediction model built on top of BERT — a transformer-based language model. The goal is to utilize issue reports, bug descriptions, and commit messages as input features and analyze their effectiveness in classifying or predicting bug-prone modules or issues.
- **Address shortcomings of traditional bug prediction models.**
  Traditional models, while lightweight and interpretable, often depend on static code metrics and historical data that are not always reliable or available. This study investigates how modern NLP-based approaches can overcome these limitations by utilizing contextual information embedded in natural language data.
- **Evaluate the proposed model against method-level approaches.**
  A key contribution of this research is the empirical comparison between the BERT-based approach and conventional method-level bug prediction models. Performance will be measured in terms of accuracy, precision, recall, and F1-score using benchmark datasets. The objective is to identify under which conditions each approach performs best and what trade-offs are involved in terms of resource usage, generalization, and practicality.

- **Contribute insights for future research and real-world applications.** In addition to empirical findings, this research aims to provide meaningful insights that can guide software engineering researchers and practitioners in selecting appropriate bug prediction methods based on their project context and goals.

Chapter 6

Literature Review

Bug prediction has been an active area of research within software engineering, with methods evolving from metric-based techniques to deep learning and natural language processing. This chapter reviews the major contributions in both traditional and modern approaches, particularly focusing on the rise of pre-trained language models like BERT and how they compare to conventional method-level strategies.

6.1 Pre-trained Language Models for Bug Prediction

Pre-trained language models, such as BERT (Bidirectional Encoder Representations from Transformers), have gained widespread adoption due to their contextual understanding of natural language. Fine-tuning BERT on domain-specific corpora, such as issue descriptions or commit messages, has shown promising results in bug classification and prediction tasks. Researchers have demonstrated that BERT-based models can capture subtle contextual clues in textual data that traditional models overlook.

Recent studies suggest that when BERT is paired with machine learning classifiers like Support Vector Machines (SVM) or Random Forest, the bug prediction performance significantly improves. The ability of BERT to understand word order and semantics provides a strong foundation for modeling the intent and severity behind bug reports, making it a compelling alternative to shallow textual models like Bag-of-Words or TF-IDF.

6.2 BERT vs. TF-IDF Feature Extraction

Traditional text vectorization techniques, particularly Term Frequency–Inverse Document Frequency (TF-IDF), have long been used in bug prediction for feature extraction. While effective to an extent, TF-IDF fails to capture the context in which words appear and treats each word independently, which limits its performance in understanding technical language in bug reports.

Comparative evaluations have consistently shown that BERT-based feature extraction outperforms TF-IDF, especially in predicting long-lived bugs. Even smaller variants of BERT, such as DistilBERT or TinyBERT, have demonstrated competitive accuracy with significantly reduced computational overhead. This makes BERT-based solutions more scalable and practical for large datasets or environments with limited resources.

6.3 Traditional Method-Level Bug Prediction

Before the emergence of NLP models, bug prediction largely relied on static and dynamic code analysis. Method-level prediction approaches utilize code metrics such as cyclomatic complexity, code churn, number of previous defects, and developer activity. These features are extracted from source code repositories and often fed into classifiers like Decision Trees, Naïve Bayes, or Random Forest.

While these models have achieved high precision and recall in several benchmark studies, they suffer from several limitations. Most notably, they rely on accurately linking bug reports to source code changes — a task that is often manual, error-prone, and time-consuming. Furthermore, these models struggle in projects with poor documentation or limited commit history. They also tend to overlook the rich semantic information present in natural language artifacts.

Despite their limitations, method-level models remain popular due to their interpretability, efficiency, and ease of deployment. However, the software engineering community is increasingly exploring hybrid models that combine code metrics with textual analysis to leverage the best of both worlds.

6.4 Gap in the Literature

While BERT and other transformer-based models are gaining traction, there is still a lack of holistic studies that directly compare them with traditional bug prediction methods using the same dataset and evaluation metrics. Most existing research either focuses solely on NLP models or solely on method-level techniques, creating a gap in understanding the trade-offs between them. This thesis aims to address this gap by conducting a side-by-side comparison, highlighting strengths, weaknesses, and areas for future improvement.

Chapter 7

Methodology

This chapter outlines the complete methodology adopted in this research, covering data collection, preprocessing, feature extraction, model training, and evaluation. The objective is to build a robust and reproducible framework to compare the performance of BERT-based bug prediction models against traditional method-level approaches.

7.1 Data Collection

The dataset used in this study was sourced from GitHub repositories via issue tracking data. Each instance in the dataset includes a title, a description (body), and a label indicating the type of issue. The distribution is as follows:

- **Training set:** 150,000 rows — each containing title, body, and label fields.
- **Test set:** 30,000 rows — containing only title and body (labels withheld for evaluation).
- **Labels:** Bug (0), Feature Request (1), and Question (2).

This distribution reflects a realistic software development environment, allowing the model to distinguish between different types of issue reports.

7.2 Data Preprocessing

To prepare the dataset for input into NLP models, we performed the following preprocessing steps:

- Combined the `title` and `body` fields to form a single textual representation per instance.
- Removed special characters, punctuation, HTML tags, URLs, and numeric tokens.
- Converted all text to lowercase and stripped unnecessary whitespace or symbols.
- Performed stratified sampling to select 2,000 rows for lightweight testing and faster iteration.

These preprocessing steps are crucial for ensuring that the model focuses on meaningful semantic content, reducing noise and improving generalization.

Epoch	Train Accuracy	Train F1	Validation Accuracy	Validation F1
1	0.72	0.70	0.79	0.78
2	0.86	0.85	0.81	0.80
3	0.93	0.93	0.80	0.80
4	0.97	0.97	0.80	0.80
5	0.98	0.98	0.80	0.80

TABLE 7.1: Training and Validation Accuracy and F1 Score over 5 Epochs

7.3 Feature Extraction

For BERT-based processing, the cleaned text was tokenized using the BERT tokenizer. A maximum sequence length was defined to truncate long sequences, and padding was applied to standardize input lengths. These tokenized inputs were converted into PyTorch tensors and passed to a DataLoader for batch processing.

7.4 Model Training

We used the ‘bert-base-uncased‘ model from HuggingFace Transformers, fine-tuned on the bug classification dataset. The training configuration is outlined below:

- **Model:** BERT-base-uncased
- **Epochs:** 5
- **Batch Size:** 12
- **Learning Rate:** 2×10^{-4}
- **Train/Validation Split:** 80/20

The model was fine-tuned using the AdamW optimizer with a learning rate scheduler. Cross-entropy loss was used due to the multiclass classification nature of the task.

7.5 Model Evaluation

To assess the model's performance, we evaluated it on the test set using standard classification metrics:

- **Accuracy:** Measures the overall correctness of predictions.
- **F1-score:** Harmonic mean of precision and recall; balances false positives and false negatives.
- **Precision:** Fraction of relevant instances among retrieved instances.
- **Recall:** Fraction of relevant instances that were retrieved.

These metrics provide a holistic view of the model's strengths and potential areas for improvement, especially in differentiating between bugs, feature requests, and questions.

Chapter 8

Results

The performance of the BERT-based bug prediction model was evaluated using standard classification metrics such as accuracy, F1-score, precision, and recall. The model demonstrated excellent convergence during training and maintained stable performance on the validation set.

The training accuracy improved steadily with each epoch, ultimately achieving 98% accuracy with an F1-score of 0.98. While the validation accuracy plateaued around 80%, the model maintained consistent generalization, avoiding overfitting. Precision and recall remained balanced throughout, indicating the model's effectiveness across all three classes: bug, feature, and question.

These results validate the capability of BERT in learning meaningful representations from issue reports, even when trained on a relatively small sample size. The outcome suggests that transformer-based models can be effectively applied to software engineering tasks with minimal feature engineering.

Chapter 9

Insights and Findings

The comparative study between NLP-based and method-level bug prediction approaches yielded several practical insights:

- **Task-specific fine-tuning enhances performance:** Fine-tuned BERT models significantly outperform generic pre-trained versions. When adapted to the bug classification task using a labeled dataset, BERT becomes highly specialized, even when trained on a small number of samples. This supports the idea that transfer learning, when correctly applied, is effective in niche domains like software issue tracking.
- **NLP-based bug prediction is practical and scalable:** One of the key advantages of NLP models like BERT is that they rely solely on textual data. This eliminates the need for complex feature engineering, such as deriving code metrics or establishing historical links between bug reports and commits. Such models are particularly useful for early-stage projects or open-source repositories with limited historical data.
- **Method-level models excel with rich project history:** In contrast, traditional method-level models demonstrate strong performance when there is sufficient historical data and well-maintained code metrics. However, they are heavily dependent on accurate traceability links between code and reported issues, which can be challenging to maintain in practice.
- **Hybrid approaches may yield better results:** The findings suggest potential for future research into hybrid models that combine the semantic understanding of NLP with the structural insights provided by method-level

metrics. Such an approach could potentially outperform either method in isolation.

- **Model choice depends on project maturity:** In mature projects with long histories and well-documented bugs, method-level models are still relevant. In contrast, for new or under-documented projects, NLP-based methods offer a viable, low-overhead alternative.

Chapter 10

Research Challenges

While implementing and fine-tuning large language models for bug prediction, several technical and practical challenges were encountered. These challenges are summarized below:

- **High computational requirements:** Training transformer-based models like BERT requires significant hardware resources, including high-performance GPUs with substantial VRAM. Limited access to such infrastructure can constrain experimentation, especially when using large datasets or complex architectures.
- **Hardware compatibility issues:** The absence of ROCm (Radeon Open Compute) support for AMD GPUs limited our ability to leverage available hardware. This made it difficult to use otherwise powerful GPUs for training, forcing reliance on cloud-based or consumer-grade alternatives, which often come with restrictions or costs.
- **Memory management and batch optimization:** Fine-tuning BERT involves large matrix operations that are memory-intensive. Determining the optimal batch size was critical to prevent out-of-memory (OOM) errors while ensuring training stability. This required careful monitoring and tuning of model parameters and data pipeline efficiency.
- **Long training time:** Even with reduced sample sizes and smaller BERT variants, training times remained relatively long, especially during hyper-parameter tuning. This created bottlenecks in experimentation and limited the number of configurations that could be tested within available time constraints.

- **Resource reproducibility and scaling:** Reproducing results across different systems with varied specifications proved to be non-trivial. Minor differences in libraries, drivers, or hardware often caused inconsistencies in model behavior, highlighting the need for robust version control and configuration management.

Chapter 11

Limitations

Despite the promising results and insights derived from this study, several limitations must be acknowledged. These limitations provide context for interpreting the findings and highlight areas for future improvement.

- **Limited computational resources:** The training and fine-tuning of transformer-based models like BERT demand significant GPU memory and processing power. Due to hardware constraints, we were unable to experiment extensively with larger architectures or perform extensive hyperparameter tuning, which may have limited the model's full potential.
- **Restricted dataset size and diversity:** Although the dataset used was real-world and representative, its relatively limited size, especially after sampling, could impact the generalizability of results. Larger and more diverse datasets could offer more robust conclusions.
- **Dependency on high-quality textual data:** NLP-based approaches like BERT heavily rely on the quality and consistency of input text. Noisy, incomplete, or ambiguous issue descriptions can reduce prediction accuracy. Open-source datasets often contain such inconsistencies.
- **Challenges with method-level feature linking:** Traditional method-level bug prediction relies on accurately linking bug reports to corresponding code changes. This process is prone to manual errors and may introduce noise into the training data, reducing the effectiveness of classical models.

- **Evaluation on limited metrics and tasks:** This study primarily focused on classification accuracy and F1-score for bug prediction. Broader evaluation metrics such as runtime efficiency, scalability, and robustness across domains were not explored due to scope limitations.
- **Model interpretability and transparency:** Transformer-based models are often considered "black boxes" due to their complex internal representations. Explaining model decisions remains a challenge, especially for debugging or integrating the model into production environments.

Chapter 12

Conclusion

This study explored and compared two prominent approaches to software bug prediction: traditional method-level models and modern NLP-based models using BERT. The investigation focused on evaluating the effectiveness, limitations, and practical implications of each technique using a real-world dataset.

A key limitation of method-level approaches lies in their dependence on accurately linking bug reports to corresponding code changes—an error-prone and labor-intensive process. These models also require well-structured historical data and extensive feature engineering, which may not be feasible in all software projects.

In contrast, BERT-based models demonstrated the ability to predict bugs using only textual data from issue reports, eliminating the need for complex preprocessing or code-based metrics. The empirical results showed that, with task-specific fine-tuning, BERT can achieve high classification performance, even on a relatively small dataset.

The literature review reinforced the growing shift toward NLP-driven solutions in software engineering, while our experiments validated the practical viability of transformer-based models in this domain. However, computational requirements remain a significant barrier to large-scale adoption of BERT, especially for researchers or developers with limited access to GPU-based infrastructure.

Overall, the study highlights the complementary nature of both approaches and opens up promising directions for hybrid models that integrate semantic understanding from NLP with structural insights from traditional methods. Future work may focus on improving the interpretability of deep learning models and evaluating performance across larger, more diverse datasets.

Bibliography

[1] Rifat Ahmed, Md. Mahbubur Rahman, and Md. Nazmul Huq. *BERT- and TF-IDF-based Feature Extraction for Long-lived Bug Prediction in FLOSS*. Journal of Systems and Software, 2023. https://www.sciencedirect.com/science/article/abs/pii/S095058492300071X

[2] Rifat Ahmed, Md. Mahbubur Rahman, and Md. Nazmul Huq. *TF-IDF-based Feature Extraction for Long-lived Bug Prediction in FLOSS*. SSRN, 2023. https://papers.ssrn.com/sol3/Delivery.cfm/615c4b45-cd7847be-a3d39450daaba67f-MECA.pdf?abstractid=4166555&mirid=1

[3] Steffen Herbold, Emanuel Giger, and Ahmed E. Hassan. *On the Performance of Method-Level Bug Prediction: A Negative Result*. Journal of Systems and Software, 2019. https://sback.it/publications/jss2019negative.pdf

[4] Unknown. *Automatic Intrinsic Bugs Classification Model using NLP and ML*. arXiv, 2024. https://arxiv.org/html/2504.01869v1

[5] Emanuel Giger, Martin Pinzger, and Harald C. Gall. *Method-Level Bug Prediction*. Proceedings of the IEEE, 2012. https://www.aau.at/wp-content/uploads/2019/11/Giger2012-methodlevel.pdf

[6] GeeksForGeeks. *Major Challenges of Natural Language Processing*. GeeksForGeeks, 2022. https://www.geeksforgeeks.org/nlp/major-challenges-of-natural-language-processing/