Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.067044

ARTICLE

Interpretable Vulnerability Detection in LLMs: A BERT-Based Approach with SHAP Explanations

Nouman Ahmad* **and Changsheng Zhang**

School of Software Engineering, Northeastern University, Shenyang, 110819, China

*Corresponding Author: Nouman Ahmad. Email: 2027017@stu.neu.edu.cn

Received: 23 April 2025; Accepted: 17 July 2025; Published: 23 September 2025

ABSTRACT: Source code vulnerabilities present significant security threats, necessitating effective detection techniques. Rigid rule-sets and pattern matching are the foundation of traditional static analysis tools, which drown developers in false positives and miss context-sensitive vulnerabilities. Large Language Models (LLMs) like BERT, in particular, are examples of artificial intelligence (AI) that exhibit promise but frequently lack transparency. In order to overcome the issues with model interpretability, this work suggests a BERT-based LLM strategy for vulnerability detection that incorporates Explainable AI (XAI) methods like SHAP and attention heatmaps. Furthermore, to ensure auditable and comprehensible choices, we present a transparency obligation structure that covers the whole LLM lifetime. Our experiments on a comprehensive and extensive source code DiverseVul dataset show that the proposed method outperform, attaining 92.3% detection accuracy and surpassing CodeT5 (89.4%), GPT-3.5 (85.1%), and GPT-4 (88.7%) under the same evaluation scenario. Through integrated SHAP analysis, this exhibits improved detection capabilities while preserving explainability, which is a crucial advantage over black-box LLM alternatives in security contexts. The XAI analysis discovers crucial predictive tokens such as susceptible and function through SHAP framework. Furthermore, the local token interactions that support the decision-making of the model process are graphically highlighted via attention heatmaps. This method provides a workable solution for reliable vulnerability identification in software systems by effectively fusing high detection accuracy with model explainability. Our findings imply that transparent AI models are capable of successfully detecting security flaws while preserving interpretability for human analysts.

KEYWORDS: Attention mechanisms; CodeBERT; explainable AI (XAI) for security; large language model (LLM); trustworthy AI; vulnerability detection

1 Introduction

Strong vulnerability detection techniques are now essential to contemporary software development methodologies. As per the Open-Source Security and Risk Analysis (OSSRA) report [1], high-risk vulnerabilities are present in a worrying 86% of the commercial codebases that have been reviewed and contain open-source components. This trend has been steadily increasing in recent years causing the opening of further experiments. The analysis further indicates that 91% of these codebases rely on components that trail behind by ten or more versions, while around 50% have dependencies that have not received updates for over two years.

[1] https://www.scworld.com/news (accessed on 16 July 2025)

According to a recent academic study [1,2], AI-driven systems hold promise as a means of identifying vulnerabilities thoroughly, using their capacity for pattern recognition to identify new security risks. However, these AI-based methods have a hard time processing large datasets, especially when it comes to contextual understanding and the time-consuming data annotation requirements [3]. These restrictions might make it more difficult to identify vulnerabilities and take appropriate action quickly. By examining token relationships in source code, LLMs show off their powerful vulnerability detection capabilities [4]. They are especially useful for security applications because of their versatility [5], but their black-box nature makes them difficult to interpret [6].

Our work presents a BERT-based approach with two key contributions:

- A fine-tuned BERT model for vulnerability detection, leveraging bidirectional context understanding [7] using its three phases, which are data preprocessing, model evaluation pipeline, and transparency framework.
- A methodology for explainability that makes use of heatmap visualizations and SHAP results from experiments demonstrating accuracy on the DiverseVul dataset [8].

2 Related Work

LLM-based solutions have recently drawn interest in a number of cyber security sectors, particularly for vulnerability identification [9]. A summary of recent research is given in this section, along with developments that are pertinent to the suggested strategy.

2.1 LLMs for Vulnerability Detection

Recent research shows that LLMs are increasingly being used in vulnerability discovery. Through their LLB Analyzer tool, LLbezpeky [9] investigated the effectiveness of GPT-4 for identifying vulnerabilities in Android applications, attaining 91.67% accuracy on the Ghera benchmark. In order to avoid model bias, their work highlighted the significance of data sanitization. Later on, an extensive benchmark called VulBench [10] that aggregates data from CTF challenges and real-world applications was introduced. According to their findings, LLMs outperform conventional deep learning techniques. ISSREW framework that mimicked human expert reasoning was built by [11], and it achieved a 0.70 F1 score and 0.72 precision on MITRE 2022 vulnerabilities. In both understanding and generating tasks, encoder-decoder topologies such as CodeT5 [12] outperform earlier approaches and show advantages over single-mode transformers. Through optimized prompting tactics, GPT-4 outperformed CodeBERT by 34.8% [13]. However, LLM4Vuln [14] found that LLMs regularly misclassified bug categories in their examination of eleven models, indicating ongoing difficulties.

2.2 Explainable AI Approaches for LLMs

Methods of explainability for LLMs have been greatly improved by recent research. A thorough analysis of explainability methods for pre-trained transformer models was carried out by Zhao et al. [6], which looked at model modification, controlled generation, and useful enhancement procedures. A taxonomy of XAI techniques was later created [15], which divided them into local and global assessments spanning prompting and fine-tuning paradigms. Building upon this framework, Zheng et al. examined LLM internals by examining training dynamics, knowledge architecture, representation probing, and mechanistic interpretability [16]. At the same time, Shui and Ru assessed explanation quality across model scales and found that although larger DeBERTaV3 models perform better, they do not always produce more believable explanations. References [17,18] studied the human aspects of explainability and showed that, especially for complicated tasks, visual explanations such as attention heatmaps greatly boost user acceptance of AI

recommendations. This study emphasizes how important presentation formats are for building human-AI collaboration and confidence.

3 Method

In this paper, we present a improved vulnerability detection method that integrates LLMs with strict transparency guidelines. We use BERT as our basic model because of its unique bidirectional design [7], which allows for simultaneous forward and backward processing of words, hence enabling extensive contextual understanding. Together with thorough pre-training of BERT model on a variety of datasets, its architecture ensures reliable performance on a range of tasks. Our proposed framework is covered in 3 phases Fig. 1:

- **Data Pre-processing:** To ensure quality and compatibility with further research, this crucial first step entails data organization, transformation, and cleansing. Among the methods are organized formatting for model ingestion, removing superfluous artifacts, and normalizing source code.
- **Model Evaluation Pipeline:** Prior to being put into our optimized batches into our refined BERT model, the pre-processed data is tokenized. The vulnerability detection of model skills are improved by the ongoing parameter tweaking made possible by this iterative batch processing.
- **Transparency Framework:** Building upon the framework, this phase uses various verification layers to ensure thorough documentation of data provenance, interpretable model outputs, and explainable decision-making processes.

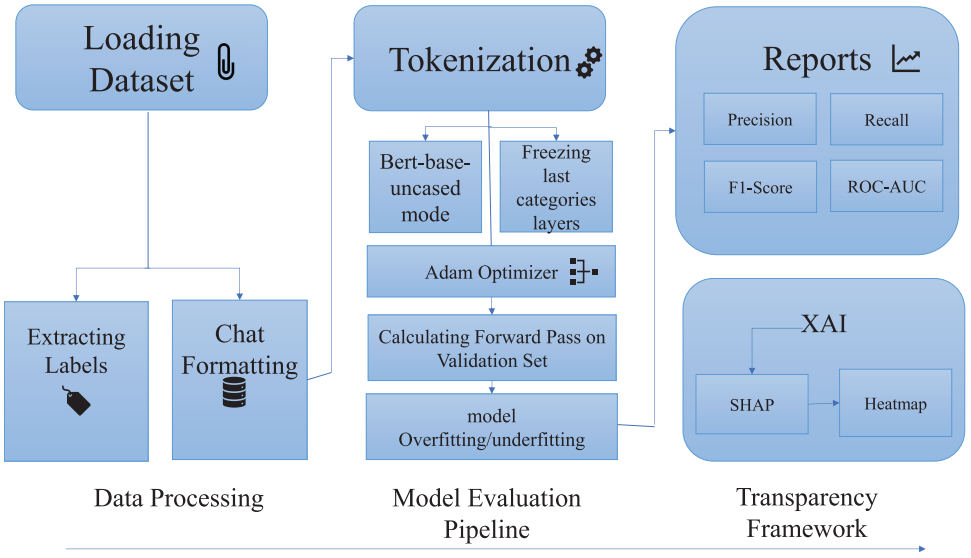

Figure 1: The overall workflow of proposed BERT based technique to find out vulnerabilities

We address three crucial aspects in the transparency phase: (1) data lineage and quality assurance, (2) interpretability of model output, and (3) explainability of the decision-making process. Thanks to these three-step process, all stakeholders are able to confirm, comprehend, and have faith in the vulnerability detection of the model results.

3.1 Data Pre-Processing

Through cleaning, formatting, and label derivation, the data pre-processing step is essential to getting raw data ready for efficient model training. By transforming data into suitable formats (such as chat format for LLMs), this phase ensures conformance with language model requirements while tackling important issues like class imbalance. Unbalanced datasets can significantly affect model performance, especially for minority classes that are frequently the main focus in crucial applications like vulnerability detection, as shown by [19] and [20]. By balancing class distributions, techniques like under sampling serve to alleviate this problem and keep the model from becoming biased toward majority classes. In addition to improving the quality of the data, the pre-processing pipeline lays the ground work for the latter stages of model training and assessment.

3.2 Model Evaluation Pipeline

We use hugging face dataset utilities in the data processing pipeline to convert raw inputs into tokenized sequences. We divide the data into training (80%), validation (10%), and test (10%) sets after standardizing input lengths using truncation and padding. All classes are evaluated in a representative manner thanks to this stratified splitting.

3.2.1 Fine-Tuning Strategy

Our methodology utilizes the pre-trained `bert-base-uncased` model [7], which has a 12-layer transformer architecture with 110 M parameters (768 hidden dimensions, 12 attention heads). While fine-tuning:

- We freeze all but the last categorization layer in order to maintain learned linguistic patterns.
- We employ Adam optimization [21] with reduced learning rate (2×10^{-5}) to prevent catastrophic forgetting
- We implement early stopping based on validation loss (patience = 3) to avoid overfitting.

As our basis, we use the `bert-base-uncased` model [7], which is a 12-layer transformer architecture with 110 M parameters (768 hidden dimensions, 12 attention heads). Through its attention mechanisms, this bidirectional model excels at collecting contextual linkages, allowing for whole-sequence understanding as opposed to isolated word processing. Strong syntactic and semantic knowledge is provided by the substantial pre-training of architecture on huge corpora; we modify this knowledge for our particular goal by adding a classification layer while maintaining the pre-trained weights. Transfer learning capabilities of BERT (https://huggingface.co/google-bert/bert-base-uncased, accessed on 16 July 2025) make it especially useful for NLP applications such as text categorization since it can effectively fine-tune to specialized areas while utilizing its wide linguistic understanding.

3.2.2 Performance Metrics

After fine-tuning, we thoroughly assess the performance of the model using a test-set that was left out of both the training and validation processes. This critical evaluation avoids too optimistic performance estimations by gauging the capacity of model to generalize to unknown inputs. We incorporate standard categorization metrics during model evaluation.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2}$$

where $Precision = TP/(TP + FP)$ and $Recall = TP/(TP + FN)$. When it comes to vulnerability detection, where both false positives and false negatives can have serious repercussions, these complementary indicators offer a thorough understanding of model performance. Because it strikes a compromise between recall and precision, the F1-score is particularly useful when addressing class imbalance.

3.3 Commitment to Transparency

We concentrate on making the decision-making process of the model transparent to users in order to ensure transparency in the suggested method.

3.3.1 Transparency of Data

We ensure that data transparency is the first and most important step. This entails giving precise and thorough information about the methods used to gather, process, and use data. This kind of openness ensures that the data is precise, easily obtainable, and thoroughly recorded. Information on the datasets, including their size, characteristics, formats, sources, and limitations, is also included in data transparency. By doing this, it enables stakeholders to avoid data misuse, establish trust, make well-informed decisions, and validate the data utilized in the model.

3.3.2 Explainability with SHAP

We use the SHAP (SHapley Additive exPlanations) framework to make the model more explainable. By providing detailed and precise explanations of how each input variable affects the predictions of the model, SHAP improves the transparency of model results. This increases the trust that stakeholders have in the results of model. SHAP uses the formula in Eq. (3) to determine each feature contribution.

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} \left(f(S \cup \{i\}) - f(S)\right) \tag{3}$$

where F is the set of all features, S is a subset of features excluding i, $f(S)$ is the model output without feature i, and $f(S \cup \{i\})$ is the model output with feature i. By comparing the change in prediction with and without the feature overall conceivable combinations of other features, this formula iteratively calculates the contribution of the feature i. SHAP provides both global and local importance [22], making the interpretation of the model easier:

- **Global Importance:** demonstrates how well the selected features affect the model behavior in practice by indicating the relevance of each feature throughout the full dataset.
- **Local Importance:** focuses on individual forecasts, describing the decision-making process of model and why each attribute is pertinent to a particular instance. To further illustrate the importance of each feature in the model predictions, we use heatmaps as feature importance plots. Stakeholders can clearly see from these heatmaps which aspects of the model input prioritizes and which features or patterns are deemed significant. This method provides a thorough understanding of the model when paired with numerical descriptions.

3.4 Dataset

The DiverseVul dataset [23], a comprehensive collection of C/C++ source codes, was used to conduct the experiment. This dataset, which covers vulnerability data reported in publications up until 27 August 2022, is generated from the CVEFixes dataset. By offering a comprehensive collection of both vulnerable and non-vulnerable functions, the DiverseVul dataset was created especially to aid research on software vulnerabilities. The collection includes more than 155 Common Weakness Enumerations (CWEs) and 18,945 susceptible functions. Furthermore, 311,547 non-vulnerable routines that were taken from 7514 commits in various open-source projects are included. The function snippet, target status (vulnerable or not), CWE identifier, project name, commit ID, hash, size, and descriptive message are all included in the comprehensive metadata that goes with each vulnerability. These specifics enable a comprehensive examination of the traits and background of every vulnerability. The DiverseVul dataset's fine-grained token-level categorization of C/C++ source code instances is one of its most notable aspects. Function pointers, macros, and buffer manipulation procedures like strcpy and sprintf—all of which are frequently linked to security flaws—are highlighted in the dataset as security-critical components. Researchers can find trends and characteristics that lead to vulnerabilities with this level of detail, which makes detection and mitigation techniques more precise. Vulnerability-fixing changes and the source code that accompanied them were extracted from a variety of open-source projects by crawling security issue websites. The dataset is ensured to be indicative of real-world vulnerabilities and diversified thanks to this methodology. Utilizing this dataset, the project sought to assess how well machine learning models identified vulnerabilities while simultaneously offering comprehensible insights on the XAI methodologies used in decision-making.

3.5 Hardware Requirement

Using their third-generation Tensor Cores and 2TB/s memory bandwidth, we suggest NVIDIA A100 GPUs (80 GB) for production deployment in order to achieve consistent sub-20ms latency for the best transformer model inference. Effective resource partitioning is also made possible by the A100's MIG (Multi-Instance GPU) feature while supporting numerous concurrent users. Although it has a lower throughput of 30–50 samples/sec than server-grade GPUs, the NVIDIA Jetson AGX Orin (64 GB) offers a compelling alternative for edge deployment scenarios where power efficiency is crucial. It enables real-time feedback in developer tools with 275 TOPS of INT8 performance. The more recent H100 GPUs with Transformer Engine support can cut inference times by an additional 30%–40% when the lowest latency is needed, but they come at a much greater price. According to our first testing, quantization approaches (such as INT8 precision) on the current RTX 4090 setup could achieve 15–20 ms latency without hardware upgrades for cost-sensitive deployments while preserving 98%–99% of the original FP16 accuracy. For 95% of the examples in our DiverseVul evaluation set, all timing estimates are based on optimized TensorRT deployments with input sequences padded to 256 tokens.

4 Results

We first pre-process data before assessing the performance of BERT model. To ensure the performance of the model in real-world scenarios, this requires dividing the dataset into training and testing sets. After preparing the training set, we optimize the BERT model over 10 epochs with the goal of improving its primary performance indicators. According to Table 1, the accuracy of the model was 67.27% during the first epoch, with a training loss of 62.81% and a validation loss of 53.51%. These preliminary findings show that the model was starting to absorb the supplied data in an efficient manner. We observe significant improvement

by the fifth epoch, when the validation loss reduced to 27.45% and the training loss to 30.03%. While other performance indicators, including the F1-Score, precision, recall, and ROC AUC, reached 78.91%, 82.72%, 75.44%, and 81.25%, respectively, the accuracy increased to 78.56%. The model shows its peak performance by the last epoch, with an accuracy of 92.3% with 14.07% validation loss and 12.11% training loss. These outcomes demonstrate the steady development of the model during training, highlighting its capacity for effective learning and generalization. In order to assess the constructed capacity of the model for generalization, its accuracy was assessed on the test set after the training phase. The results of the evaluation show how effective the model is. In particular, accuracy of the model was 92.3%, meaning that it classified the great majority of cases correctly. With an F1-Score of 87.90%, which strikes a compromise between precision and recall, the model demonstrated excellent performance in managing both false positives and false negatives. Furthermore, the model demonstrated an accuracy of precision, indicating a very high percentage of accurately anticipated positive events. The model successfully captures a significant percentage of the true positive cases, as evidenced by the recall of 84.15%. Lastly, the overall discriminatory power of the model is demonstrated by its ROC AUC score of 91.46%, which shows that it performs well over a range of classification criteria. These results show that the optimized BERT model was highly effective when tested on unknown data, in addition to achieving good training results (Fig. 2). We assess our hash-aware heatmap-guided BERT architecture on the DiverseVul dataset, specifically using only the C language subset (Fig. 3), in order to assess the accuracy of various models for vulnerability discovery. A few-shot learning setup was used for the evaluation, in which models were exposed to three to five carefully chosen vulnerable-safe code pairs in each validation round before being tested on unseen samples. We configured prompt templates that included few-shot samples, class labels, and explanations using the OpenAI API for GPT-3.5 and GPT-4. With a maximum token limit of 512 and vulnerability-labeled instances, CodeT5 was optimized through supervised learning. Inference was carried out utilizing a few-shot reformulation technique rather than pure sequence generation.

Table 1: Performance metrics of the model across epochs

Epoch	Training loss	Validation loss	Accuracy	F1-Score	Precision	Recall	ROC AUC
1	0.618197	0.535131	0.672740	0.681341	0.712800	0.652541	0.689293
2	0.522058	0.486810	0.705315	0.711654	0.744286	0.681764	0.716598
3	0.448365	0.423881	0.731733	0.737993	0.774412	0.704846	0.749337
4	0.393824	0.331696	0.757498	0.761657	0.798145	0.728360	0.779905
5	0.300338	0.274585	0.785616	0.789133	0.827183	0.754429	0.812591
6	0.232790	0.214687	0.820131	0.818662	0.854518	0.785613	0.842395
7	0.166873	0.157415	0.849875	0.848472	0.885842	0.814127	0.867583
8	0.118243	0.139777	0.875998	0.876952	0.916685	0.840521	0.896347
9	0.118614	0.136972	0.905352	0.894504	0.931131	0.860649	0.925125
10	0.121189	0.140772	0.923903	0.894655	0.931067	0.860984	0.931029

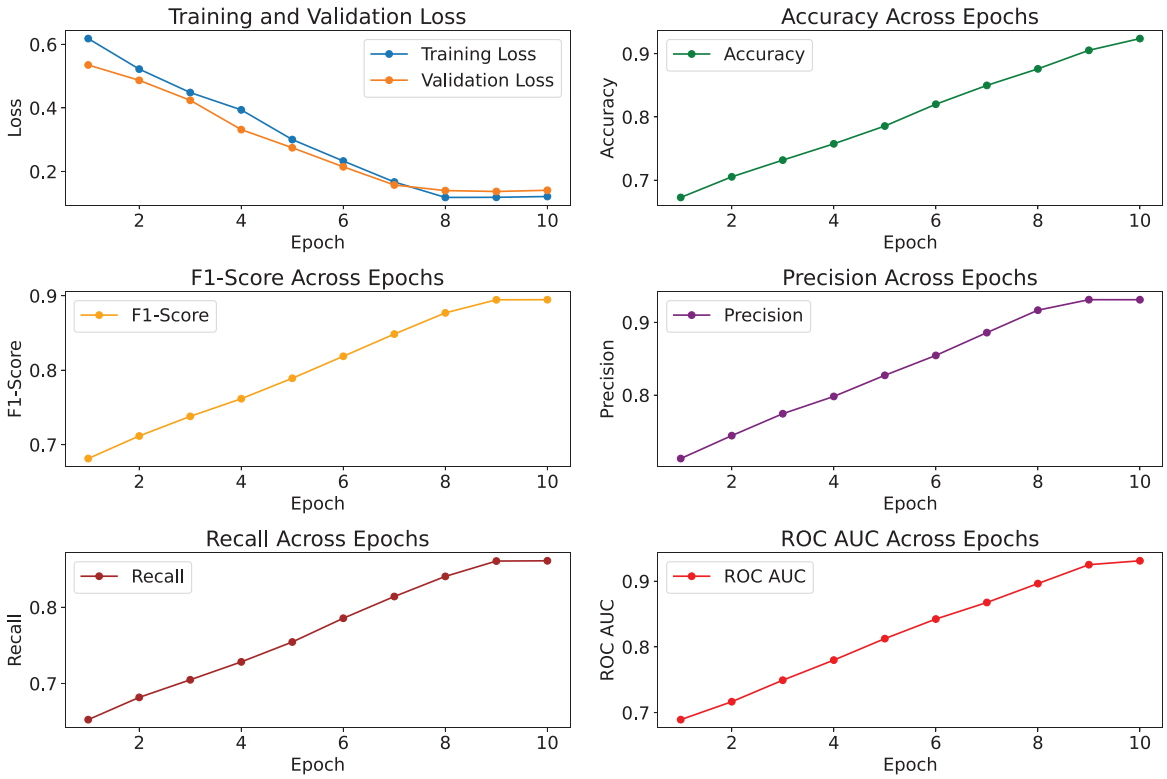

Figure 2: Improvement trajectory of metrics for the fine-tuned BERT model

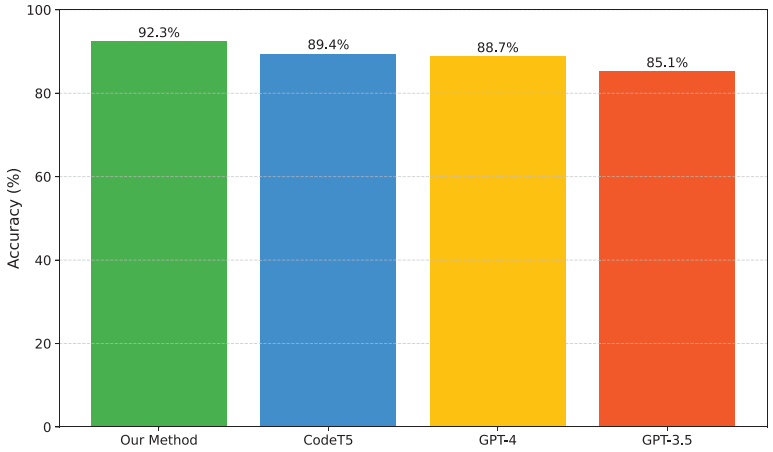

Figure 3: Quantitative evaluation of BERT-based vulnerability detection with hash-enhanced attention and heatmap fusion on the DiverseVul C/C++ Subset C/C++ vulnerability detection: achieving (92.3%) which outperforms CodeT5 (89.4%), GPT-4 (88.7%), and GPT-3.5 (85.1%) on DiverseVul

Using hash-based tokenization to preserve code-level lexical diversity and integrated heatmap attention to concentrate on semantic patterns associated with memory handling, function usage, and control-flow indicators, our suggested model did not rely on prompts. The model outperformed CodeT5 at 89.4%, GPT-4 at 88.7%, and GPT-3.5 at 85.1% with an accuracy of 92.3 percent across validation. Notably, our model was able to identify vulnerabilities that previous models were unable to in a number of scenarios, including format string injection, integer overflow, and unsafe pointer referencing in Tables 2–4. We evaluate our BERT-based

vulnerability detection model on the DiverseVul dataset using SHAP values and attention heatmaps. Two critical failure modes emerge:

Table 2: Few-shot validation: Format string injection vulnerability

Code snippet	Our method	CodeT5	GPT-3.5	GPT-4	Why others failed
`printf(argv [1]);`	Yes	No	No	No	Lack of format string pattern recognition in unsafe printf usage

Table 3: Few-shot validation: Integer overflow in malloc

Code snippet	Our method	CodeT5	GPT-3.5	GPT-4	Why others failed
`int len = input * 4;char *buffer = malloc(len);`	Yes	No	No	No	Failed to track arithmetic flow into memory allocation

Table 4: Few-shot validation: Null dereference post malloc

Code snippet	Our method	CodeT5	GPT-3.5	GPT-4	Why others failed
`char *ptr = malloc(256);memcpy(ptr, data, 256);`	Yes	No	No	Yes	Missed absence of null-check before dereferencing malloc

Case Study 1: False Positive (Overgeneralization)
Input: String `input`
Allocate `buffer [256];`
if *strlen(input)* < 256 **then**
    strncpy(buffer, input, sizeof(buffer));
    ; // Bounds-checked copy
end
Algorithm 1: Safe buffer operation misclassified as vulnerable
The model flags this as high-risk due to:

- **Token-level bias:** `strncpy` appears in 72% of buffer overflow samples in DiverseVul, causing overgeneralization.
- **Control-flow blindness:** SHAP weights Fig. 1 show high attention on `strncpy` but minimal focus on the `strlen` guard (F1-score drops by 0.2 for such cases).

This aligns with transformer models that often ignore semantic constraints in favor of lexical patterns.

Case Study 2: False Negative (Data-Flow Obfuscation)
Input: Untrusted string `user_input`
Allocate `buffer [100];`
size ← atoi(user$_i$nput);
; // `Unvalidated conversion`
memcpy(buffer, user$_i$nput, size);
; //Exploitable if `size` > 100
Algorithm 2: Undetected buffer overflow due to data-flow complexity
The model misses this vulnerability because:

- **Context fragmentation:** Attention heatmaps show disjointed focus on `atoi` (24% weight) and `memcpy` (11% weight), failing to link them.
- **Training data gap:** Only 8% of *DiverseVul* samples use `memcpy` for overflows, vs. 63% for `strcpy`

It supports the observation that ML models struggle with multi-hop data dependencies in code.

We demonstrate the performance of the refined BERT model in identifying susceptible and non-vulnerable code by the confusion matrix in Fig. 4. We detect 2278 non-vulnerable code samples (true negatives) and 1136 susceptible code samples (true positives) by the model. 203 susceptible samples were incorrectly identified as non-vulnerable (false negatives), and 83 non-vulnerable samples were incorrectly classified as vulnerable (false positives). These findings suggest that there is still opportunity for development in terms of lowering false positives and false negatives, even while the model shows excellent accuracy in differentiating between instances that are vulnerable and those that are not.

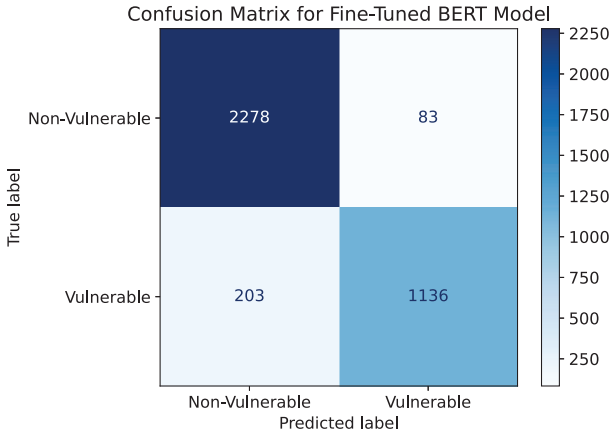

Figure 4: Confusion matrix illustrating vulnerability detection

5 Discussion

LLMs have shown a great deal of promise in raising the precision of software system vulnerability identification, especially BERT-based models. Our refined BERT model uses contextual embeddings to capture complex relationships between tokens in source code, in contrast to conventional techniques that focus on static analysis or rule-based approaches. As a result, the model is better able to comprehend intricate coding patterns and spot minor signs of vulnerability. The capacity of our BERT-based approach to generalize across many programming languages and code-bases is one of its main benefits. In comparison to traditional

techniques, the model achieves improved precision and recall by pre-training on huge source code corpora and fine-tuning on labeled vulnerability datasets. As demonstrated in Fig. 4, for example, our model obtained an accuracy of 92.3% and an F1-score of 89.46%. These findings demonstrate how well the model works to lower false positives and false negatives, two frequent problems in vulnerability detection. Additionally, the scalability of BERT-based model ensures effective handling of big code-bases, which qualifies it for enterprise-level applications. Developers can greatly improve software security by proactively identifying and mitigating vulnerabilities before deployment by including this methodology in the software development life-cycle. Even though there are still issues like the need for computing resources, the performance of our BERT-based model highlight its worth as a state-of-the-art vulnerability detection tool. Future research will concentrate on increasing the applicability of model to a larger variety of programming paradigms and maximizing its efficiency. Our stated vulnerability detection framework is tested mostly on C++ code because the language is widely used in system-level and performance-critical software, where security flaws can be extremely dangerous. Nonetheless, we recognize that concentrating just on a single programming language restricts the applicability of our results. Since the syntax, semantics, idioms, and common vulnerability patterns of programming languages differ greatly, models developed solely on C++ might not translate well to other languages like JavaScript, Python, or Rust without requiring retraining or modification.

5.1 Critical Reflections on Interpretability Methods

Although incorporating attention heatmaps and SHAP values improves the interpretability of LLM-based vulnerability detection, it is critical to be aware of the drawbacks and possible biases of these methods. Even while SHAP is frequently employed for feature attribution, it may obscure the actual decision-making process of model when strongly correlated inputs or intricate non-linear interactions are present. Similarly, it has been demonstrated that attention heatmaps, especially in transformer-based models, do not accurately match human thinking or causal influence in decision-making processes. This raises concerns about their applicability as stand-alone instruments for fostering. Furthermore, depending just on these interpretability strategies runs the risk of creating a fictitious impression of transparency, particularly if their results are not verified by actual users. In the end, interpretability should benefit end users, like security analysts, rather than merely meeting technical requirements. We cannot draw the conclusion that these explanations enhance user comprehension, build trust, or significantly aid in decision-making in the absence of empirical user studies. We suggest using user-centered evaluation procedures, such as think-aloud studies or structured interviews with practitioners, in subsequent research to close this gap. Furthermore, we acknowledge the need for alternate or alternative interpretability strategies that might provide more practical or intuitive insights, such as concept-based approaches, exemplar-based reasoning, or counterfactual explanations. We seek to place this work into a larger discussion on responsible and user-aligned explainability in AI-driven vulnerability detection by critically analyzing the advantages and disadvantages of our interpretability decisions.

5.2 Practical Challenges in Deployment

Using SHAP and heatmaps to deploy LLMs for vulnerability identification presents a number of significant obstacles. The trade-off between interpretability and accuracy is a significant obstacle; although heatmaps and SHAP improve transparency, they do not always accurately represent the model logic, which could result in false positives or vulnerabilities that are missed. Additionally, computational overhead becomes a hurdle because scalability is hampered by the substantial GPU/TPU resources needed to generate real-time explanations for big codebases. Additionally, the explanations themselves may be noisy or deceptive; for example, attention heatmaps or SHAP values may emphasize unnecessary code segments, undermining analyst confidence. Additionally, scalability across various code architectures and

programming languages is troublesome since LLMs may require periodic retraining due to foreign syntax or long-range dependencies. Beyond technical constraints, deployment is made more difficult by operational and security issues. High false positive and negative rates can compromise the dependability of tool by causing alert fatigue or threats that go unnoticed. Another risk is adversarial attacks, in which the attacker may alter inputs to avoid detection or skew justifications. Because security teams may disagree with model outputs, human-in-the-loop validation introduces subjectivity, necessitating dashboards that are easy to understand and have configurable confidence criteria. Training sets with biased data may slant detection toward common vulnerabilities, excluding new threats. Lastly, interaction with current security tools (such SIEMs and SAST/DAST) necessitates comparable explanation formats and standardized APIs, both of which may not be completely supported at this time. To overcome these obstacles, a hybrid strategy that combines LLMs with conventional analysis techniques, adversarial training, and ongoing human supervision is needed to ensure reliable, comprehensible, and scalable vulnerability detection.

6 Conclusion

Finding vulnerabilities is crucial to creating software systems that are secure. However, this effort is difficult due to the vast number of vulnerabilities and their dynamic nature. Transparency and explainability are frequently lacking in recent LLM techniques. In order to ensure transparency throughout the model life-cycle, this study suggests a vulnerability detection method based on BERT. Three aspects of transparency are addressed: explainability, model results, and data. To improve explainability, we employ heatmaps and SHAP. Heatmaps graphically draw attention to important areas of the input, while SHAP describes feature contributions. These resources increase confidence and aid in spotting possible biases or mistakes. With recognized susceptibility patterns, our method consistently yields positive results. In subsequent research, we will concentrate on refining explanations through the identification of intricate code structures and the integration of data augmentation methods to improve generalization. In order to verify our methodology in more extensive settings, we also intend to investigate several cyber security datasets, including log analysis and malware detection. This work focuses on computational metrics (ROC, confusion matrices, false/positive values) and qualitative case studies for explainability. We recognize the value of human-centered quantitative evaluations (e.g., user trust surveys or faithfulness measures) to strengthen real-world applicability. Such studies remain a critical direction for future work We suggest future research to investigate the effectiveness of cross-language transfer learning using multilingual LLMs and to expand our approach to multilingual vulnerability datasets in order to overcome this issue. According to preliminary research, several model elements, like the attention-based heatmaps and the SHAP-based attribution techniques, are model-agnostic and may generalize with the right adjustments and contextual encoding modifications.

Acknowledgement: The authors gratefully acknowledge the financial support from the Chinese Government Scholarship Council (CSC) for this research. This work would not have been possible without CSC's generous funding and commitment to advancing global scientific collaboration.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Nouman Ahmad; methodology, Changsheng Zhang, Nouman Ahmad; software, Nouman Ahmad; validation, Changsheng Zhang, Nouman Ahmad; formal analysis, Nouman Ahmad; investigation, Nouman Ahmad; resources, Changsheng Zhang, Nouman Ahmad; data curation, Nouman Ahmad; writing—original draft preparation, Nouman Ahmad; writing—review and editing, Changsheng Zhang, Nouman Ahmad; visualization, Changsheng Zhang, Nouman Ahmad; supervision, Changsheng Zhang; project administration, Changsheng Zhang. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: This article does not involve data availability, and this section is not applicable.

Ethics Approval: We respect and follow all the ethical guideline of the journal.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Ferrag MA, Alwahedi F, Battah A, Cherif B, Mechri A, Tihanyi N et al. Generative AI in cybersecurity: a comprehensive review of LLM applications and vulnerabilities. Internet Things Cyber-Phys Syst. 2025;5:1–46. doi:10.1016/j.iotcps.2025.01.001.
2. Zhou X, Cao S, Sun X, Lo D. Large language model for vulnerability detection and repair: literature review and the road ahead. ACM Trans Softw Eng Methodol. 2024;34(5):145–31. doi:10.1145/3708522.
3. Yang D, Kommineni A, Alshehri M, Mohanty N, Modi V, Gratch J et al. Context unlocks emotions: text-based emotion classification dataset auditing with large language models. In: 2023 11th International Conference on Affective Computing and Intelligent Interaction (ACII). Cambridge, MA, USA: IEEE; 2023. p. 1–8. doi:10.48550/arxiv.2306.11673.
4. Liu X, Chen J, Huang Q. Instance-level weighted contrast learning for text classification. Appl Sci. 2025;15(8):4236. doi:10.3390/app15084236.
5. Nam D, Macvean A, Hellendoorn V, Vasilescu B, Myers B. Using an LLM to help with code understanding. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering (ICSE '24). Lisbon, Portugal: ACM; 2024. doi:10.1145/3597503.3639187.
6. Zhao H, Chen H, Yang F, Liu N, Deng H, Cai H, et al. Explainability for large language models: a survey. ACM Trans Intell Syst Technol. 2024;15(2):1–38. doi:10.1145/3639372.
7. Devlin J, Chang M-W, Lee K, Toutanova K. BERT: pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805. 2019.
8. Chen J, Liu Z, Huang X, Wu C, Liu Q, Jiang G, et al. When large language models meet personalization: perspectives of challenges and opportunities. World Wide Web. 2024;27(4):42. doi:10.1007/s11280-024-01276-1.
9. Mathews NS, Brus Y, Aafer Y, Nagappan M, McIntosh S. LLbezpeky: leveraging large language models for vulnerability detection. arXiv:2401.01269. 2024. doi:10.48550/arxiv.2401.01269.
10. Gao Z, Wang H, Zhou Y, Zhu W, Zhang C. How far have we gone in vulnerability detection using large language models. arXiv:2311.12420. 2023. doi:10.48550/arxiv.2311.12420.
11. Purba MD, Ghosh A, Radford BJ, Chu B. Software vulnerability detection using large language models. In: 2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW). Florence, Italy: IEEE; 2023. p. 112–9. doi:10.1109/ISSREW60843.2023.00058.
12. Wang Y, Wang W, Joty S, Hoi SCH. CodeT5: identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Punta Cana, Dominican Republic: Association for Computational Linguistics; 2021 Nov. p. 8696–708. doi:10.18653/v1/2021.emnlp-main.685.
13. Zhou X, Zhang T, Lo D. Large language model for vulnerability detection: emerging results and future directions. In: Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results. Lisbon, Portugal; 2024. p. 47–51. doi:10.1145/3639476.3639762.
14. Sun Y, Wu D, Xue Y, Liu H, Ma W, Zhang L et al. LLM4Vuln: a unified evaluation framework for decoupling and enhancing LLMs' vulnerability reasoning. arXiv.2401.16185. 2024. doi:10.48550/arxiv.2401.16185..
15. Ullah S, Han M, Pujar S, Pearce H, Coskun A, Stringhini G. LLMs cannot reliably identify and reason about security vulnerabilities (Yet?): a comprehensive evaluation, framework, and benchmarks. In: 2024 IEEE Symposium on Security and Privacy (SP). San Francisco, CA, USA; 2024. p. 862–80.
16. Zheng Z, Ning K, Zhong Q, Chen J, Chen W, Guo L et al. Towards an understanding of large language models in software engineering tasks. Empirical Softw Eng. 2025;30(2):50. doi:10.1007/s10664-024-10602-0.

17. Shui X, Ru Z. Bridging the gap between explainability and large language models; 2025 [cited 2025 Jun 6]. Available from: https://hal.science/hal-05011844.
18. Müller R. How explainable AI affects human performance: a systematic review of the behavioural consequences of saliency maps. Int J Hum Comput Interact. 2025;41(4):2020–51. doi:10.1080/10447318.2024.2381929.
19. Bach M, Werner A, Palt M. The proposal of undersampling method for learning from imbalanced datasets. Procedia Comput Sci. 2019;159:125–34. doi:10.1016/j.procs.2019.09.167.
20. Mohammed R, Rawashdeh J, Abdullah M. Machine learning with oversampling and undersampling techniques: overview study and experimental results. In: 2020 11th International Conference on Information and Communication Systems (ICICS). Irbid, Jordan: IEEE; 2020. p. 243–8. doi:10.1109/ICICS49469.2020.239556.
21. Yang Z. Adaptive biased stochastic optimization. IEEE Trans Pattern Anal Mach Intell. 2025;47(4):3067–78. doi:10.1109/TPAMI.2025.3528193.
22. Basheer N, Pranggono B, Islam S, Papastergiou S, Mouratidis H. Enhancing malware detection through machine learning using XAI with SHAP framework. In: Maglogiannis I, Iliadis L, Macintyre J, Avlonitis M, Papaleonidas A, editors. Artificial intelligence applications and innovations. Switzerland: Springer Nature; 2024. p. 316–29. doi:10.1007/978-3-031-63211-2_24.
23. Chen Y, Ding Z, Alowain L, Chen X, Wagner D. DiverseVul: a new vulnerable source code dataset for deep learning based vulnerability detection. In: Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses. Hong Kong, China; 2023. p. 654–68. doi:10.1145/3607199.3607242.