LLM-Enhance: Fine-Tuning Large Language Models for Enhanced Detection of Common Weakness Enumerations

Shrey Modi
Dept. of Computer Eng. & Computer Sci.
California State University, Long Beach
Long Beach, CA, United States
shreydharmendra.modi01@student.csulb.edu

Rahul Vishwakarma
Dept. of Computer Eng. & Computer Sci.
California State University, Long Beach
Long Beach, CA, United States
rahuldeo.vishwakarma01@student.csulb.edu

Amin Rezaei
Dept. of Computer Eng. & Computer Sci.
California State University, Long Beach
Long Beach, CA, United States
amin.rezaei@csulb.edu

***Abstract*—Detecting Common Weakness Enumerations (CWEs) in computational systems remains a challenge for hardware and software developers. In this paper, we scrutinize the efficacy of machine learning algorithms in addressing the above issue, employing both real and synthetic data generated via data augmentation techniques, generative adversarial networks, and rule-based synthesis. Furthermore, we propose fine-tuning Large Language Models (LLMs) to generate two high-quality CWE datasets of 50,000 entries each. We then deploy traditional machine learning algorithms such as support vector classifier, random forest, and naive Bayes, as well as advanced methods such as convolutional neural networks, graph neural networks, and transformer-based architectures, on the curated dataset to perform the task of CWE detection. Results show that fine-tuned LLMs and transformer architectures outperform others, demonstrating their effectiveness in identifying vulnerabilities across hardware-software boundaries.**

***Index Terms*—Common Weakness Enumeration; Machine Learning; Large Language Models; Vulnerability Detection**

I. Introduction

Common Weakness Enumerations (CWEs) are critical vulnerabilities in hardware and software systems [1]. This paper presents a Machine Learning (ML) and Large Language Model (LLM)-based framework for CWE detection using both real and synthetic data. We construct two large CWE datasets enhanced via data augmentation and LLM fine-tuning, and evaluate different ML models on these datasets.

Recent studies emphasize the advantages of ML over traditional static analysis for early vulnerability detection, including applications in Internet of Things (IoT) [2] and log-based anomaly detection [3]. LLMs have shown promise in extracting vulnerability information from unstructured sources [4], zero-shot vulnerability repair [5], code vulnerability detection [6], policy-based protection [7], and hardware security assertion generation [8]. However, challenges remain, such as prompt sensitivity and incomplete responses. Despite these limitations, LLMs demonstrate strong potential in automating security tasks, including penetration testing [9] and solving offensive security challenges [10].

Fig. 1 illustrates the selected CWEs that lie at the intersection of software and hardware vulnerabilities, providing context for the scope of our dataset and analysis.

II. LLM-Enhance

In this section, we propose **LLM-Enhance**, a comprehensive workflow designed for hardware and software vulnerability detection shown in Fig. 2. First, we gather and augment CWE data. The collected data is then fine-tuned with LLMs to create two distinct datasets with thousands of data points. These enhanced datasets then undergo vulnerability detection using both traditional ML techniques and state-of-the-art advanced architectures.

A. Data Generation

In the initial phase of our research, we collect approximately 2,270 data points related to five CWEs including CWE-212, CWE-226, CWE-311, CWE-319 and CWE-459 from the DiverseVul [11] and NIST SARD [12] datasets. We then expand the dataset to 5,000 high-quality entries using advanced synthetic data generation tools such as Gretel [13], CTGAN [14], and PALETTE [15].

Data transformation enhances compatibility across environments. For Llama [16] and Falcon [17], this involves converting function source code and reinterpreting vulnerabilities as CSV instructions. Mistral [18] requires preserving columnar structure in JSONL format, while StarCoder [19] focuses on structuring raw code into syntax trees and dependency graphs.

Pre-trained LLMs based on the transformer architecture [20] are initially trained on massive datasets, enabling them to perform a wide range of language tasks such as translation,

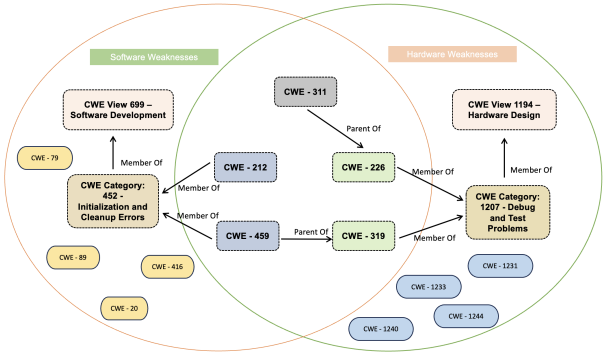


Fig. 1: CWEs at the Intersection of Software and Hardware Weaknesses

979-8-3315-5193-3/26/$31.00 ©2026 IEEE

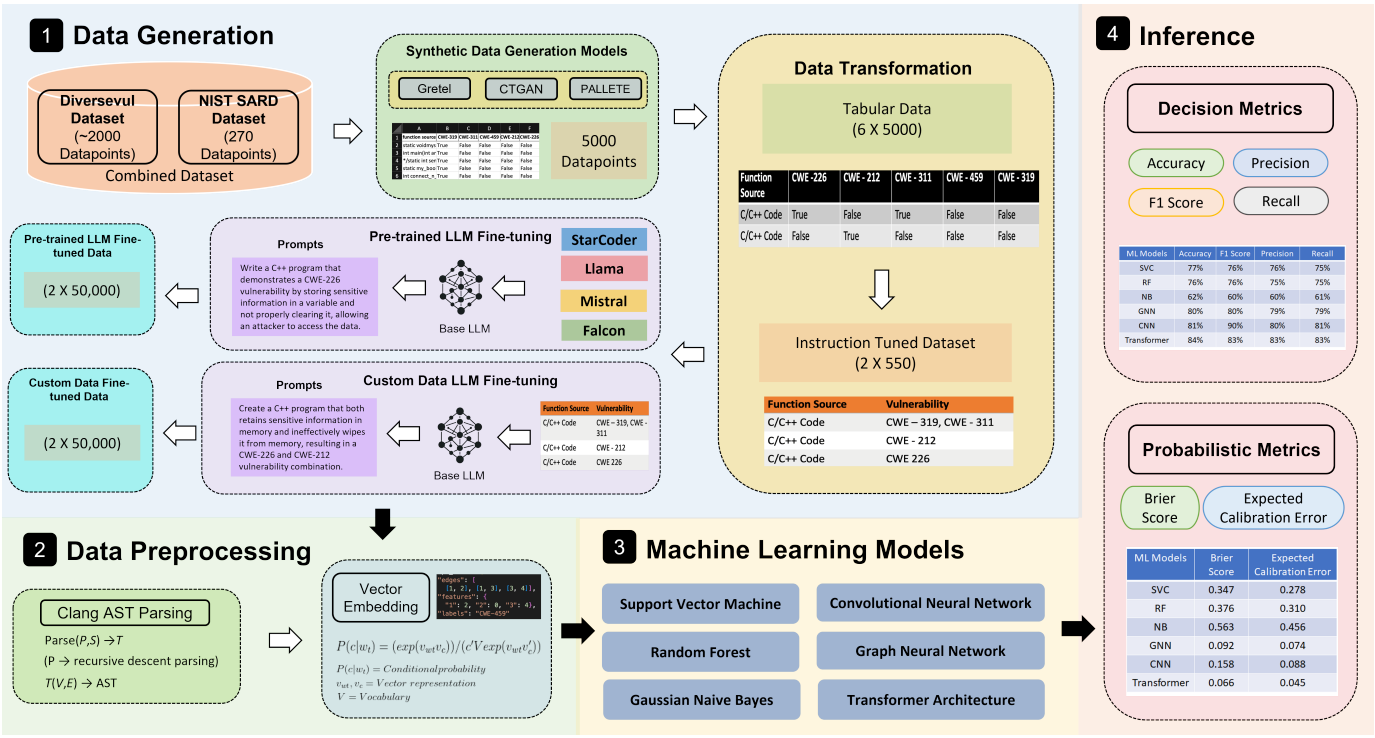


Fig. 2: LLM-Enhance Workflow

summarization, and question answering. Despite their broad capabilities, these models often require fine-tuning for domain-specific applications. The goal of fine-tuning is to minimize cross-entropy loss [21], defined as:

$$\text{Loss} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \quad (1)$$

Where N is the number of samples, y_i is the actual label, and $\hat{y}_i$ is the predicted probability. Parameter updates are implemented via gradient descent, optimized by the Adam optimizer [22]:

$$\theta_{t+1} = \theta_t - \eta \cdot \text{Adam}(\nabla_\theta \text{Loss}) \quad (2)$$

Here, η represents the learning rate. Through this fine-tuning process, the LLM not only adapts to the specifics of hardware and software vulnerabilities but also enhances its prediction accuracy.

After the data fine-tuning is done, we generate a dataset of about 50,000 data points for each dataset (i.e., the custom LLM dataset and the pre-trained LLM dataset).

B. Data Preprocessing

After the datasets have been generated from the fine-tuned LLM models, we transform C++ code snippets from the function source column into Abstract Syntax Trees (ASTs) using Clang, a specialized compiler front end for the C language family. These ASTs provide a structured and comprehensive analysis of the code, making it easier to identify and interpret various programming constructs. After generating ASTs, the next step involves converting them into JSON format. JSON's flexibility, manageability, and lightweight structure simplify our preprocessing workflow and enhance the depth and relevance of subsequent analyses.

C. Machine Learning Algorithms

We evaluate six machine learning algorithms on our curated datasets to compare their effectiveness in vulnerability detection.

Support Vector Classifier (SVC) [23] separates classes by constructing a hyperplane that maximizes the margin. It uses kernel functions and regularization to handle complex classification tasks. Random Forest (RF) [24] builds an ensemble of decision trees trained on random subsets of data. It improves robustness and accuracy by aggregating predictions from multiple trees. Gaussian Naive Bayes (GNB) [25] is suitable for high-dimensional data. It applies Bayes' theorem with Gaussian assumptions to estimate class probabilities based on feature distributions. Convolutional Neural Networks (CNNs) [26] are effective at detecting patterns in structured data such as source code. Our CNN uses multiple convolutional and pooling layers followed by dense layers and dropout for regularization. Graph Neural Networks (GNNs) [27] model relationships in graph-structured data. They use message passing between nodes to learn complex dependencies, making them ideal for vulnerability detection in software and hardware systems. Transformers like BERT [28] use self-attention mechanisms to capture bidirectional context in text. Pre-trained on large corpora, they are fine-tuned for specific tasks such as classification and question answering.

TABLE I: Decision Metrics for Pre Trained LLM Dataset

ML Models	Accuracy	F1 Score	Precision	Recall
SVC	46%	43%	43%	44%
RF	44%	43%	43%	42%
GNB	37%	36%	36%	35%
GNN	48%	45%	46%	47%
CNN	47%	45%	46%	46%
Transformer	53%	53%	53%	52%

TABLE II: Decision Metrics for Custom LLM Dataset

ML Models	Accuracy	F1 Score	Precision	Recall
SVC	77%	76%	76%	75%
RF	76%	76%	75%	75%
GNB	62%	60%	60%	61%
GNN	80%	80%	79%	79%
CNN	81%	80%	80%	81%
Transformer	84%	83%	83%	83%

III. Experimental Results

To evaluate the performance of different ML models on the curated pre-trained and custom fine-tuned datasets, we consider both decision and probabilistic metrics. Each dataset is split into 70% training, 15% validation, and 15% testing to ensure balanced evaluation and robust training. Our training minimizes cross-entropy loss with the Adam optimizer, using a learning rate (η) of 0.001 over 30 epochs. The experiments are run on a 12-core Intel Xeon Silver 4310 processor with 16GB of DDR4 RAM and an NVIDIA 64GB A16 GPU.

A. Decision Metrics

Tables I and II show the decision metrics (i.e., accuracy, F1 score, precision, and recall) for pre-trained and custom fine-tuned LLM datasets respectively. The results reflect the evolution in ML models, highlighting the advancements in performance and architecture. In the first comparison, traditional models like SVC and RF showcased moderate performance. GNB, however, demonstrated a noticeably weaker ability to handle the classification task. More advanced models, including GNNs and CNNs, displayed some improvement, but the transformer architecture clearly outperformed them, establishing a new benchmark for model performance. The second comparison highlights substantial improvements across all models, revealing the superiority of the custom LLM fine-tuning. While we observe almost the same trend in the performance of ML models, CNN shows higher performance compared with GNN in this dataset. The transformer architecture, however, took center stage, excelling across all decision metrics and firmly establishing its superiority.

One notable point is that once we switch Llama 2 to Llama 3, we observe 3% increase in accuracy, from 81% to 84%. Llama 3 utilizes a larger and more diverse training dataset, expanding its understanding of language nuances and context, thereby enhancing its generalization capabilities. Additionally, advanced fine-tuning methods help Llama 3 adapt better to specific tasks.

TABLE III: Probabilistic Metrics for Pre-Trained LLM Dataset

ML Models	Brier Score	Expected Calibration Error
SVC	0.456	0.347
RF	0.487	0.391
GNB	0.620	0.510
GNN	0.342	0.287
CNN	0.320	0.236
Transformer	0.290	0.187

TABLE IV: Probabilistic Metrics for Custom LLM Dataset

ML Models	Brier Score	Expected Calibration Error
SVC	0.347	0.278
RF	0.376	0.310
GNB	0.563	0.456
GNN	0.092	0.074
CNN	0.158	0.088
Transformer	0.066	0.045

B. Probabilistic Metrics

The Brier Score (BS) and Expected Calibration Error (ECE) provide realistic perspectives on the performance of classification models. The BS calculates the squared discrepancies between the predicted probabilities and the actual results, offering a numerical representation of probabilistic prediction accuracy. A model with a lower BS suggests that its probability estimates are closely aligned with the actual outcomes, aiding in evaluating not just the binary correctness but also the confidence of the predictions. In addition, ECE evaluates the extent to which the predicted probabilities mirror the actual probabilities of outcomes. It scrutinizes the calibration of a model's predictions, ensuring that the probabilities accurately correspond to the actual event likelihoods. A model with BS and ECE near zero is considered well-calibrated, meaning its predicted probabilities closely match actual outcomes. These metrics matter most when probability quality is more important than accuracy alone. The formula for BS calculation is as follows:

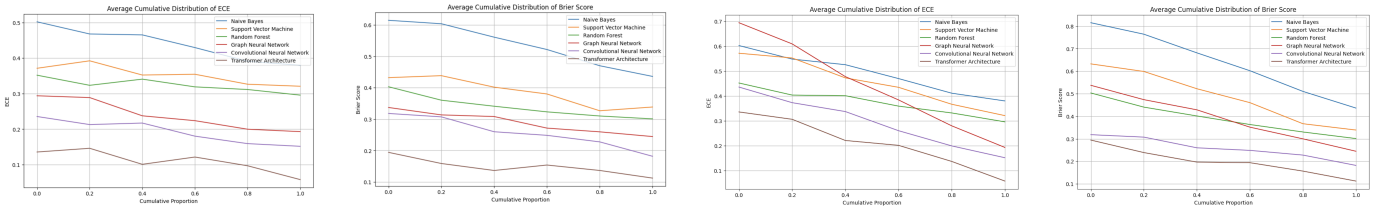
$$\text{BS} = \frac{1}{N}\sum_{i=1}^{N}(f_i - o_i)^2 \tag{3}$$

Where f_i denotes the predicted probability for an event, o_i indicates the actual outcome (1 or 0), and N is the total number of predictions. The formula for ECE calculation is as follows:

$$\text{ECE} = \sum_{m=1}^{M}\frac{|B_m|}{n}\left|\text{acc}(B_m) - \text{conf}(B_m)\right| \tag{4}$$

Where n is the total number of samples, B_m is the set of indices of samples in bin m, acc(B_m) is the accuracy within bin m, and conf(B_m) is the mean predicted probability in bin m.

Tables III and IV evaluates the probabilistic metrics (i.e., BS and ECE) for pre-trained and custom fine-tuned LLM dataset respectively. Deep learning models outperform traditional methods, with transformer architecture achieving the lowest BS (0.066) and ECE (0.045) in custom fine-tuned LLM dataset. In contrast, traditional models like GNB exhibit the highest BS (0.563) and ECE (0.456) in custom fine-tuned LLM



(a) Average Cumulative Distribution of ECE for Pre-trained LLM Dataset

(b) Average Cumulative Distribution of BS for Pre-trained LLM Dataset

(c) Average Cumulative Distribution of ECE for Custom LLM Dataset

(d) Average Cumulative Distribution of BS for Custom LLM Dataset

Fig. 3: Performance Analysis of Different ML Models

dataset, reflecting poor probabilistic accuracy and calibration. Comparing the results from the two tables, it is evident that transformer architecture demonstrates superior probabilistic accuracy and calibration, consistently achieving the lowest BS and ECE metrics. In addition, all the models have better probabilistic metrics once trained on the custom LLM dataset compared with the pre-trained LLM dataset.

In addition, Fig. 3 display the cumulative distribution of BS and ECE for different ML models, demonstrating decreasing trends from GNB to transformer architecture. Both scores decrease progressively across models, indicating improved prediction accuracy and calibration in more sophisticated models, particularly deep learning ones. This trend highlights the enhanced predictive capabilities of models like CNNs and transformer architectures due to their ability to capture complex patterns in data.

IV. Conclusion

In this paper, we presented **LLM-Enhance**, an ML-based vulnerability detection approach that leverages LLM-generated datasets. We created two datasets of 50,000 hardware–software CWEs using real samples, synthetic augmentation, and LLM fine-tuning. We then evaluated six ML models for CWE detection. While traditional models such as SVC, RF, and GNB achieve moderate accuracy, advanced architectures such as GNN and CNN perform better, and transformer models deliver the strongest overall results. We also highlighted the value of probabilistic metrics for assessing prediction reliability.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Award No. 2245247.

References

[1] Common weakness enumeration: Root cause mapping of vulnerabilities. In *https://cwe.mitre.org/*.

[2] Rakibul Hassan, Charan Bandi, Meng-Tien Tsai, Shahriar Golchin, Sai Manoj P D, Setareh Rafatirad, and Soheil Salehi. Automated supervised topic modeling framework for hardware weaknesses. In *24th International Symposium on Quality Electronic Design (ISQED)*, pages 1–8, 2023.

[3] Van-Hoang Le and Hongyu Zhang. Log-based anomaly detection with deep learning: How far are ww? In *Proceedings of the 44th International Conference on Software Engineering (ICSE)*, page 1356–1367, 2022.

[4] Virendra Ashiwal, Soeren Finster, and Abdallah Dawoud. Llm-based vulnerability sourcing from unstructured data. In *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 634–641, 2024.

[5] Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. Examining zero-shot vulnerability repair with large language models. In *IEEE Symposium on Security and Privacy (SP)*, pages 2339–2356, 2023.

[6] Jin Wang, Zishan Huang, Hengli Liu, Nianyi Yang, and Yinhao Xiao. Defecthunter: A novel llm-driven boosted-conformer-based code vulnerability detection mechanism. In *arXiv, 2309.15324*, 2023.

[7] Sudipta Paria, Aritra Dasgupta, and Swarup Bhunia. Divas: An llm-based end-to-end framework for soc security analysis and policy-based protection. In *arXiv, 2308.06932*, 2023.

[8] Rahul Kande, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Shailja Thakur, Ramesh Karri, and Jeyavijayan Rajendran. (security) assertions by large language models. In *IEEE Transactions on Information Forensics and Security*, volume 19, pages 4374–4389, 2024.

[9] Jiaxin Yu, Peng Liang, Yujia Fu, Amjed Tahir, Mojtaba Shahin, Chong Wang, and Yangxiao Cai. An insight into security code review with llms: Capabilities, obstacles, and influential factors. In *arXiv, 2401.16310*, 2025.

[10] Minghao Shao, Boyuan Chen, Sofija Jancheska, Brendan Dolan-Gavitt, Siddharth Garg, Ramesh Karri, and Muhammad Shafique. An empirical evaluation of llms for solving offensive security challenges, 2024.

[11] Yizheng Chen, Zhoujie Ding, Lamya Alowain, Xinyun Chen, and David Wagner. 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 (RAID)*, page 654–668, 2023.

[12] Paul E Black. A software assurance reference dataset: Thousands of programs with known bugs. In *Journal of Research of the National Institute of Standards and Technology*, volume 123, pages 1–3, 2018.

[13] Ainul Haezah Noruzman, Norjihan Abdul Ghani, and Nor Saradatul Akmar Zulkifli. Gretel.ai: Open-source artificial intelligence tool to generate new synthetic data. In *Journal of Innovation in Enigneering and Applied Social Sciences*, volume 1, pages 15–22, 2021.

[14] Virginia Cortes, José González, Gustau Camps-Valls, Alec Radford, and Yoshua Bengio. Adaptive pruning of neural language models. In *Advances in Neural Information Processing Systems*, pages 6598–6607, 2019.

[15] Rahul Vishwakarma and Amin Rezaei. Risk-aware and explainable framework for ensuring guaranteed coverage in evolving hardware trojan detection. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9, 2023.

[16] Hugo Touvron et al. Llama: Open and efficient foundation language models. In *arXiv, 2302.13971*, 2023.

[17] Guilherme Penedo et al. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data, and web data only. In *arXiv, 2306.01116*, 2023.

[18] Albert Q. Jiang et al. Mistral 7b. In *arXiv, 2310.06825*, 2023.

[19] Raymond Li et al. Starcoder: May the source be with you! In *arXiv, 2305.06161*, 2023.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, volume 30, 2017.

[21] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in Neural Information Processing Systems (NIPS)*, volume 31, 2018.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *arXiv, 1412.6980*, 2017.

[23] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, volume 20, pages 273–297. Springer, 1995.

[24] Leo Breiman. Random forests. In *Machine learning*, volume 45, pages 5–32. Springer, 2001.

[25] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, volume 3, pages 41–46, 2001.

[26] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324. IEEE, 1998.

[27] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. In *IEEE Transactions on Neural Networks*, volume 20, pages 61–80, 2009.

[28] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *arXiv, 1810.04805*, 2018.