Knowing your weaknesses is your greatest strength: Mapping CVE to CWE by leveraging CWE Hierarchy and fine-tuned LLMs

Stefano Simonetto
University of Twente
Enschede, The Netherlands
s.simonetto@utwente.nl

Ronan Oostveen
University of Twente
Enschede, The Netherlands
r.oostveen@utwente.nl

Thijs van Ede
University of Twente
Enschede, The Netherlands
t.s.vanede@utwente.nl

Peter Bosch
University of Twente
Enschede, The Netherlands
h.g.p.bosch@utwente.nl

Willem Jonker
University of Twente
Enschede, The Netherlands
w.jonker@utwente.nl

Abstract

Effective defense against threat actors requires that security professionals accurately identify the underlying weaknesses associated with common vulnerabilities and exposures (CVEs). This understanding is crucial for deploying appropriate defensive mechanisms and prioritizing remediation efforts. However, manually mapping CVEs to common weakness enumerations (CWEs) has become increasingly impractical due to the rapid increase of new CVEs and the extensive, complex CWE taxonomy. In 2025, the number of CVEs awaiting analysis exceeded 25,000.

To automate the mapping between CVEs and CWEs, we propose to leverage two insights. To harness the power of large language models, we first fine-tune different language models to perform this mapping based on the vulnerability-to-weakness relation. Second, we propose a supervised framework leveraging the hierarchical structure of CWEs, where we first categorize vulnerabilities into broad CWE classes (e.g., Injection, Buffer Overflow), which helps capture high-level patterns, and then utilizes specialized subnetworks to distinguish fine-grained differences within each class.

Evaluated on a benchmark that covers 95% of all CVEs associated with a CWE, our approach improves F1-score by 5% over the best prior supervised method, demonstrating the value of combining model fine-tuning with hierarchy-aware classification.

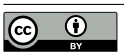
CCS Concepts

• **Security and privacy → Software and application security**; *Software security engineering*.

Keywords

Vulnerability, mapping, weakness, fine-tuning, hierarchy

ACM Reference Format:
Stefano Simonetto, Ronan Oostveen, Thijs van Ede, Peter Bosch, and Willem Jonker. 2026. Knowing your weaknesses is your greatest strength: Mapping CVE to CWE by leveraging CWE Hierarchy and fine-tuned LLMs. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '26), June 1–5, 2026, Bangalore, India.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3779208.3785376



ASIA CCS '26, Bangalore, India

ACM ISBN 979-8-4007-2356-8/26/06
https://doi.org/10.1145/3779208.3785376

1 Introduction

New vulnerabilities that pose risks to organizations are continuously discovered in an ever-evolving security landscape. To systematically track and catalog these vulnerabilities, NIST established the Common Vulnerabilities and Exposures (CVE) [10], which documents publicly disclosed cybersecurity threats with concrete details such as product specifics, version numbers, and vendor information. However, while CVEs provide essential technical details, they often lack the broader contextual information needed for a comprehensive understanding of their potential impact on systems.

To bridge this gap, NIST categorizes vulnerabilities under the Common Weakness Enumeration (CWE) [40], which offers a more abstract and structured classification by highlighting the underlying weaknesses. This abstraction is further enriched by the National Vulnerability Database (NVD) [29], which not only aggregates CVE information but also includes mappings to the corresponding CWEs. These mappings help security professionals contextualize vulnerabilities, assess their potential impact, and prioritize remediation efforts.

For instance, beginning in 2025, an average of 130 new CVEs are added to the database each day [30], with every entry requiring manual processing by CVE Numbering Authorities (CNAs). This labor-intensive process is compounded by the complexity of the CWE taxonomy, which demands careful selection from a broad range of categories. As shown in Figure 6 in the Appendix, experts struggle to assign appropriate CWE labels to new CVEs promptly. Accurately labeling CWEs is essential as it enables the clustering of related CVEs and provides valuable insights into the patterns exploited by threat actors. Recent advances in natural language processing (NLP), particularly through transformer-based models like BERT, RoBERTa, and GPT, have shown promise in addressing complex text understanding tasks. These models excel at capturing intricate linguistic patterns and have begun to demonstrate potential in automating the task of mapping CVEs to CWEs.

In this work, we address the scalability challenges of manually mapping CVEs to CWEs by proposing an automated solution that leverages large language models (LLMs) alongside the hierarchical structure of the CWE taxonomy. Our method first employs fine-tuning to translate CVEs into their corresponding CWEs, and then

exploits the parent-child relationships within the CWE framework, which is an aspect that has been underutilized by previous automated methods. By doing so, our method enhances the effectiveness of vulnerability categorization.

2 Motivation and contribution

The ever-increasing volume of reported CVEs has outpaced manual review: in 2025, over 55% of CVEs remained unvetted by NIST experts as of August 2025 (Table 10 in the Appendix), and each entry requires, on average, more than 8 days to receive a corresponding CWE. At the same time, empirical studies show a strong link between specific CWEs and exploited vulnerabilities, making accurate CWE labeling critical for prioritizing remediation. For example, the CISA Known Exploited Vulnerabilities (KEV) database reports that CWE-787 (Out-of-bounds Write) accounts for 11% of exploited vulnerabilities, underscoring how CWE information correlates with real-world risk.

Yet assigning the correct CWE to each CVE is nontrivial: vulnerability descriptions vary widely in length and format, critical details are often omitted, and the CWE taxonomy itself is large and hierarchical. Automating this mapping can reduce per-CVE analysis time from days to seconds, allowing security teams to quickly triage new disclosures and allocate resources accordingly [37].

To address these challenges, we propose a supervised, two-step neural-network pipeline that fine-tunes LLM embeddings on CVE descriptions and leverages the CWE hierarchy for coarse and fine-grained classification. The contributions of this paper are twofold, yielding a 5% increase in F1-score compared to prior supervised methods:

(1) We systematically fine-tuned and compared a wide range of language models, such as BERT, RoBERTa, Phi, GPT, and LLaMA, specifically tailored for the CVE-to-CWE mapping task, revealing new performance insights and outperforming existing supervised, unsupervised, and LLM-based approaches (see Section 7).
(2) We introduce a two-step approach that leverages NIST's labeling and the inherent hierarchical structure of the CWE taxonomy to optimize the CVE-to-CWE mapping process. This approach not only enhances the mapping accuracy by exploiting parent-child relationships within the CWE framework but also determines the appropriate granularity, whether to assign a parent or a child CWE, to improve the precision of vulnerability classification (see Section 8).

The code is made available for further improvements and to foster collaboration in this field at [5].

3 Background

In this section, we review the features and structures of CVEs and CWEs, discuss how NIST assigns CWEs to CVEs, and introduce the role of LLMs in transforming text descriptions into vector representations.

3.1 CVEs and CWEs

Common Vulnerabilities and Exposures (CVEs) are distinct labels attributed to publicly acknowledged cybersecurity vulnerabilities. These labels play a crucial role in enabling security practitioners and organizations to discuss specific vulnerabilities using standardized terminology. However, CVE entries often contain complex technical details, such as product names and versions, and their syntax varies across years, posing challenges for conventional NLP techniques. Common Weakness Enumeration (CWEs) represent a community-developed list of prevalent software weaknesses and security vulnerabilities. In contrast to CVEs, which pinpoint individual vulnerabilities, CWEs classify broader categories of weaknesses, encompassing various manifestations of similar vulnerabilities. This systematic categorization enhances comprehension of underlying vulnerability causes, thereby enabling the implementation of more comprehensive security measures throughout the software development and system deployment phases. CWEs detail the conditions that facilitate vulnerability exploitation, the underlying procedures, and the potential impact, thereby assisting in threat assessment and mitigation [2].

The structure of CWEs is inherently hierarchical, reflecting a generalization-specialization relationship as noted in [20]. In this hierarchy, higher-level categories represent more abstract weaknesses, while lower-level categories provide more specific descriptions of vulnerability types (see Figure 1).

To effectively automate the mapping of CVEs to CWEs, it is essential to grasp the structure and evolution of these frameworks. For our purposes, we follow the CWE representation adopted by NIST experts, which utilizes only a specific subset of CWEs. This selective approach avoids the broader categorical mappings suggested by CWE-MITRE and instead focuses on the more granular levels: Pillars, Classes, Bases, and Variants, as detailed in Table 1.

Table 1: CWE framework grouping: categories that are deprecated or not used in NIST mappings, and categories actively used for mapping.

Category	A CWE entry representing a collection of related weaknesses that share a common characteristic.
No-info	A weakness with insufficient information in its description to be accurately mapped.
Other	A weakness that falls outside the scope of the categories adopted by NIST.
Not available	No mapping is provided for the corresponding CVE entry.
Pillar	The most general type of weakness that cannot be abstracted further.
Class	A high-level weakness described in abstract terms, independent of specific languages or technologies.
Base	A weakness described in abstract terms but with enough details to infer specific detection and prevention methods.
Variant	A weakness tied to specific products or technologies, typically involving particular languages or platforms.

3.2 Language models

3.2.1 BERT and RoBERTa. BERT (Bidirectional Encoder Representations from Transformers) [14] is a neural network method based on transformers designed for natural language processing. It learns

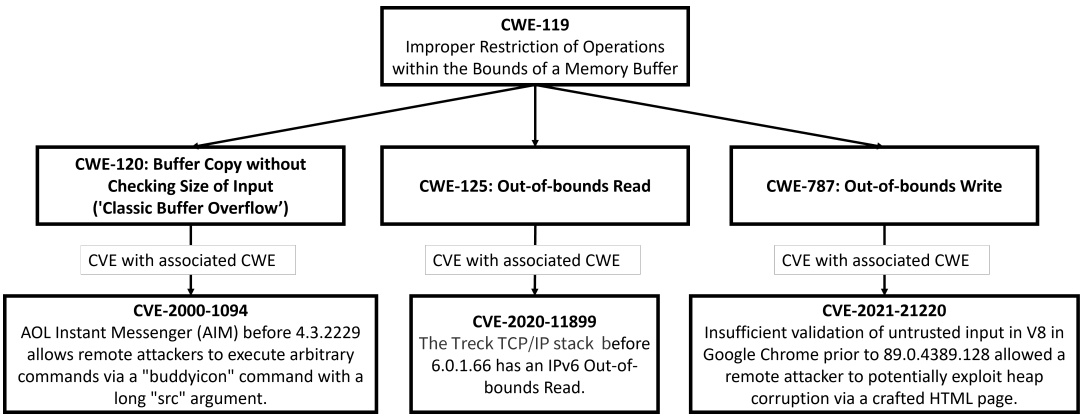


Figure 1: Hierarchical CWE schema with associated CVEs examples

from both directions, encompassing the entire word sequence in a sentence or query during training. This allows BERT to learn from the surrounding words and understand the context of a sentence or query, leading to state-of-the-art performance in various NLP tasks. BERT employs transformers [41] that utilize an attention mechanism to understand the contextual relationships among words and subwords within a sequence. In our scenario, this capability can facilitate CWE prediction based on CVE text. A derivative of BERT, a robustly optimized version with modifications in the tokenizer and the network architecture and ignored NSP tasks during training, is called RoBERTa [26].

3.2.2 Large Language Models. While BERT functions as an embedding-based model suited for classification tasks, Large Language Models such as GPT (Generative Pretrained Transformers) [19] operate as a decoder-based generative model.

The transformer architecture used in GPT is a significant advancement over previous approaches to NLP, such as RNN and CNN. By applying multi-headed self-attention over input context tokens followed by position-wise feedforward layers, GPT generates an output distribution over target tokens [35]. This design allows GPT to excel in natural language understanding tasks, including entity recognition and relationship extraction.

Moreover, several open-weight models—such as LLaMA [16], Gemini [15], and Mistral [23]—are now achieving performance comparable to GPT. In our context, these LLMs are valuable for predicting the correct CWE based on CVE descriptions.

3.2.3 SLMs. The quality of training data has long been recognized as a crucial factor in model performance [28]. Small Language Models (SLMs), such as Phi, build on this understanding by emphasizing the importance of high-quality data, as advocated in "Textbooks Are All You Need" [18]. The training data for Phi includes a mixture of synthetic datasets specifically designed to enhance commonsense reasoning and general knowledge across diverse domains, including science. As SLMs continue to demonstrate strong performance despite their compact size, they are gaining momentum as key components in the development of agentic AI systems [7].

3.2.4 Embedding. In this work, we extract embeddings from various language models to obtain vector representations that capture the semantic meaning of CVE descriptions. This approach ensures a "level playing field," allowing us to easily substitute different language models within our pipeline, as they produce a standardized output format that can be further processed consistently.

4 Related Work

Mapping CVEs to their corresponding CWEs is a challenging task that has inspired several NLP-based approaches. Prior work has explored a range of techniques, from classical machine learning with simple text features to deep learning with embeddings. However, existing methods have notable limitations, and there remains a need for a more comprehensive approach that leverages state-of-the-art generative models and the CWE hierarchical taxonomy to improve mapping accuracy.

The authors of [38] embed CVEs and CWEs into a shared latent space and assign CWEs based on cosine similarity between their embeddings. Although conceptually appealing, this embedding-based matching approach performs poorly in practice.

Methods such as those presented in [6], [2], [42], and [32] address the issue through diverse sentence embedding techniques. In [6], the approach utilizes Bag of Words (BoW) as the vectorizer, followed by a feature selection stage. The selected features are then fed into a random forest classifier for CWE prediction, focusing only on 19 CWE-IDs. In [36] the authors use Term Frequency-Inverse Document Frequency (TF-IDF) based feature vector and Support Vector Machine (SVM) technique to map CVEs to CWEs. The targeted CWE set is composed of only 6 CWEs without considering the CWE hierarchy.

Threatzoom [2] leverages the CWE hierarchy and utilizes (TF-IDF) as a vectorizer and employs hierarchical decision-making based on whether the CWE predicted is a leaf or not, iterating the process until a leaf is reached; however, the performance on the test set is low with an accuracy of 32% on the NVD dataset. Similarly, the work in [33] classifies security patches into fine-grained vulnerability types, utilizing the CWE hierarchy to navigate as deeply as possible within the structure. However, this approach may not always yield optimal results, as the deepest classification

does not necessarily equate to better mapping. The authors of V2W-BERT [12] introduce a transformer-based multi-label hierarchical classification method that organizes 124 weaknesses into three layers for vulnerability classification. This work marks the first integration of a transformer-based approach with the CWE hierarchy. Their method is limited to predicting the existence of links between CWEs and does not adopt the CWE-1003 perspective.

While a different approach is proposed by [24] based on hierarchical cross-encoders, their method focuses on capturing both the semantic similarity between CVE descriptions and CWE texts. Their pipeline trains separate cross-encoders for each level of the MITRE hierarchy, supplemented by a binary classifier to route predictions between levels. Reported performance is high with an overall accuracy of 72.1% on 130 CWE classes. However, we do not include this method in our empirical comparison due to data leakage between the training, validation, and test sets. Although the source code is not publicly available, our analysis of their released dataset splits revealed overlapping CVEs across these sets. This leakage makes a direct comparison with our results unreliable.

Recent studies have increasingly investigated the use of generative AI models for cybersecurity tasks. Notably, works such as [25] and [11] explore prompt-engineered and zero-shot methods leveraging models like GPT to map CVEs to CWEs. Although these approaches show considerable promise, they face limitations in output controllability. Specifically, while large language models can return responses in structured formats, it remains challenging to encode the full CWE hierarchy and view-specific constraints directly into the prompt. This hinders the ability to adapt model outputs to domain-specific requirements and practical application contexts.

4.1 State of the Art

Recent work by [42] and [32] explores neural architectures for mapping CVEs to CWEs, both proposing a TextCNN–BiGRU classification network. [42] adopts a hybrid approach that combines an enhanced TF-IDF weighting scheme with Word2Vec embeddings, while [32] employs BiLSTM for embedding generation. In both cases, the resulting embeddings are passed to a TextCNN–BiGRU classifier, which uses convolutional layers to capture local n-gram patterns and bidirectional GRUs to model sequential dependencies.

[42]'s framework focuses on the 18 most frequently occurring CWE types and treats them as mutually exclusive classes. [32] builds on a similar architecture but reports marginally improved performance through different embedding choices.

While these methods advance vulnerability classification by integrating neural architectures and an improved weighting strategy, they still rely on relatively conventional embedding techniques and consider the CWE taxonomy as a flat label space, thereby overlooking hierarchical relationships and contextual dependencies among weakness categories.

4.2 Gap in the literature

Neither of the papers described in Section 4 employs the state-of-the-art open-weight generative transformer architecture such as LLaMA, which is the focus of this work. Moreover, none of the approaches, whether open or closed source, utilize a language model fine-tuned specifically for the CVE to CWE mapping task.

Crucially, while some methods incorporate the CWE hierarchy, none propose a tailored neural network architecture for each CWE family to effectively distinguish parent categories from their children. Instead, current models, such as [2], map CVEs to the lowest node in the CWE hierarchy. This rigid strategy fails to account for scenarios in which a higher-level (parent) CWE more accurately reflects the described vulnerability, indicating the need for a more flexible, hierarchy-aware approach.

5 Preliminary Analysis

The CVE-CWE dataset plays a pivotal role in addressing the challenges identified in related work, particularly in managing data imbalance and variability in vulnerability descriptions.

As shown in Table 10 in the Appendix, the number of published CVEs has steadily increased over the years. In 2023, over 37% of all vulnerabilities were classified under just three CWEs, highlighting a significant imbalance in the data. Notably, the top three CWEs each year account for more than 28% of the total, indicating that over a quarter of vulnerabilities fall within these categories. Among the most frequent CWEs, two families consistently dominate: "Buffer Overflow" and "Injection." An interesting outlier is CWE-310 (Cryptographic Issues), which accounted for 21.07% of cases in 2014, a year notably impacted by the Heartbleed vulnerability, which is a critical flaw in the OpenSSL cryptographic library[1].

5.1 Dataset

A robust dataset is essential for training any supervised model, and for CVE-to-CWE mapping, the quality and balance of data are especially critical. In this study, we adopted a supervised learning strategy, as previous work has shown this to be the most effective approach for this task. However, our efforts to build the dataset revealed inherent challenges, particularly data imbalance, which we further discuss in Section 10.

One significant challenge is the variability in the specificity of CWE labels. Some labels are overly generic or even deprecated, which can obscure the true root cause of a CVE, as discussed in Section 3.1. To ensure a reliable foundation, we constructed our dataset by obtaining CVE entries and utilizing only the NIST-approved mappings from CVEs to CWEs, as specified by "Weaknesses for Simplified Mapping of Published Vulnerabilities"[2] framework.

The following subsections detail the steps taken to prepare the dataset, starting from NIST-approved labels, including converting multi-label instances, applying a minimum sample threshold, and creating training, validation, and test splits.

5.1.1 From multi-label to single label. Although the CWE framework allows a CVE to carry multiple weakness labels, in practice, only 1.5% of all CVEs in the NVD-derived dataset have more than one CWE. Instead, we follow prior NLP practice by duplicating each multi-label CVE into as many examples as it has CWEs, assigning exactly one CWE per copy. This transformation preserves complete label fidelity. Since multi-label CVEs constitute only 1.5% of the

[1] https://nvd.nist.gov/vuln/detail/cve-2014-0160
[2] https://cwe.mitre.org/data/definitions/1003.html

data, this approach increases our total instance count by under 2% (to 225886 entries), without affecting the overall class distribution. For training and validation, we utilize the enlarged dataset, while for testing, we ensure that the predicted CVEs have originally only one label associated with them.

5.1.2 Minimum Sample Threshold. As reviewed in Section 4, supervised approaches in the literature have adopted different thresholds, which are the minimum number of CVEs that are related to a CWE. Based on different thresholds, different amounts of classes are taken into account, ranging from 6 CWEs [36] to 18 CWEs [42], typically selecting the most prevalent CWEs. In our study, we adopted a threshold that covers 95% of all CVEs associated with a CWE, resulting in the inclusion of 57 distinct CWEs. The justification for this selection and its inherent limitations are further examined in Section 10.

5.1.3 Train-Validation-Test Split. We reserved 10% of the dataset, restricted to the 57 selected classes, as a hold-out test set (12,845 samples), preserving the original class distribution. From the remaining 90% of entries, we constructed the training and validation sets. To mitigate imbalance (for example, CWE-79 originally exceeded 25,000 entries), we applied undersampling to cap each class at 3,000 samples, yielding a balanced pool of 72,372 CVE descriptions. This pool was then split 90% for training and 10% for validation. We ensured that the test set is completely disjoint from both the training and validation sets.

Figure 7 in the Appendix visually summarizes the CWE distribution, highlighting the adopted thresholds, the training-validation and test split, and the CWEs that were excluded.

6 Fine-tuning and model selection methodology

In this section, we describe our process for selecting the most suitable language model for predicting the correct CWE from CVE descriptions. Building on our prepared dataset, we evaluate baseline performances and apply targeted fine-tuning to enhance the models' grasp of cybersecurity-specific terminology and structure.

To determine the optimal model for the CVE-to-CWE mapping task, we experimented with several state-of-the-art language models. Our evaluations include classification models such as BERT, BERT Large, and RoBERTa, as well as autoregressive models like Phi and LLaMA. The model selection process, illustrated in Figure 2, begins with data acquisition (step 0) and follows a series of steps detailed in the following subsections.

6.1 Dataset split and NLP preprocessing

The training, validation, and test datasets are the ones described in Section 5.1.

A1. NLP To assess the influence of natural language characteristics on CVE description classification, we applied a standard NLP pipeline comprising tokenization, lowercasing, stop-word removal, lemmatization, and stemming. This preprocessing enabled us to evaluate how these NLP techniques impact the classification performance. For a concrete example, please refer to Figure 8 in Appendix E, which illustrates a CVE description before and after the NLP processing.

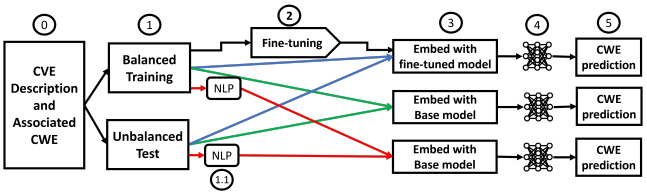


Figure 2: Model comparison pipeline, with colors highlighting the datasets used for training and testing. The red sections emphasize data processed by the NLP pipeline and embedded using only the base model, while the green sections highlight the CVE descriptions directly embedded by the base model. In blue, the training and testing CVE descriptions are fed into the fine-tuned model. Note that for the fine-tuning process, we exclusively use the balanced dataset.

6.2 Fine-tuning

Language models often struggle with less common entities [27], a limitation that is especially pronounced in the cybersecurity domain. When deploying models in areas with limited or highly specialized data, customization through fine-tuning becomes essential to achieve optimal performance [39]. The core idea in fine-tuning for CVE-to-CWE mapping is to expose a model to a sufficient number of vulnerability descriptions associated with the same CWE, thereby enabling it to internalize domain-specific terminology and patterns.

Cybersecurity terminology in everyday language is either infrequently used (e.g., ransomware, API, OAuth, exfiltrate, keylogger) or interpreted variably across contexts (e.g., honeypot, patch, handshake, virus), as highlighted in [1]. Fine-tuning allows language models to better align with these nuances and follow instructions more precisely [31]. Unlike the RAG-based approach employed in [34], we choose to fine-tune our models, ensuring that all necessary information is directly embedded within the model itself.

We utilize the following prompt for fine-tuning across all language models, as exemplified by the instance of CVE-2010-1459 below:

> You are a helpful cybersecurity expert designed to help me matching Common Vulnerability and Exposure (CVE) to Common Weakness Enumerations (CWE).
> ### **CVE description**:
> The NeXTDecode function in tif_next.c in LibTIFF allows remote attackers to cause a denial of service (out-of-bounds write) via a crafted TIFF image, as demonstrated by libtiff5.tif.
> ### **Predicted CWE**:
> 787

For fine-tuning, we employed LoRA (Low-Rank Adaptation of Large Language Models) [21], a technique that introduces trainable low-rank matrices into each layer of the transformer architecture while keeping the original pre-trained weights frozen. This significantly reduces the number of trainable parameters (modifying less than 6.5% of the model) while maintaining performance. Details on the language model parameter counts and configurations for each setup are provided in Table 12 in Appendix F.1, and a more

comprehensive explanation of the LoRA method is available in Appendix F.

We evaluated multiple LoRA configurations across various language models. The corresponding validation loss curves for each configuration are presented in Appendix F.1.

During the fine-tuning of all models, a significant decrease in validation loss was observed in the initial epochs. While the loss continued to decline in subsequent epochs, the reduction became negligible towards the final epochs, indicating diminishing returns and suggesting that additional epochs are unlikely to yield further significant improvement. The energy consumption related to the fine-tuning of the different language models is discussed in Table 14 in Appendix F.1.

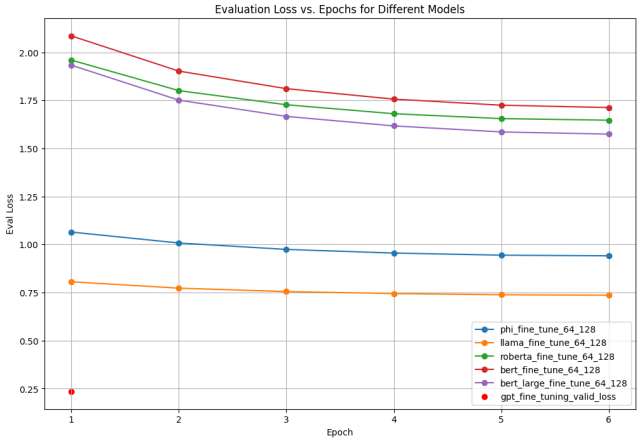


Figure 3: Validation loss for each model during fine-tuning

6.3 Embedding

Given that our selected language models generate outputs in different formats (e.g., GPT produces text while BERT outputs probability distributions), we extract a uniform output format to ensure a fair comparison. To enhance the robustness of our pipeline and mitigate issues such as hallucination [22], we extract the embeddings from the final layer of each model instead of relying on direct CWE prediction. These embeddings, which vary in dimensionality according to the model architecture, are then passed to a downstream neural network that learns to map the CVE description to its corresponding CWE.

We generate embeddings for CVE descriptions using both the base and fine-tuned versions of each model. For the base models, we experiment with embeddings derived from both the original CVE descriptions and those processed through the NLP pipeline described in Section 6.1.

The selection of embeddings is critical, as it varies depending on the model architecture. For classification models such as BERT and RoBERTa, experimental results from [13] indicate that MEAN-pooling yields the best sentence representation compared to CLS, MEAN, and MAX pooling strategies. In contrast, for autoregressive models like Phi and LLaMA, which predict the next word based on preceding tokens, we extract embeddings only from the last token in the CVE description, aligning with their sequential prediction mechanism.

6.4 Training

To ensure a fair comparison among models, we performed a comprehensive grid search to identify the optimal hyperparameters for each one. During each training session, we explored critical hyperparameters, including the number of hidden layers, the number of neurons per layer, and the batch size. The number of epochs was controlled via a callback that monitored the F1-weighted score on the validation set, halting training when the current score surpassed all previous scores. The boxplots 13a 14b 14a in Appendix illustrate the distribution of validation F1-scores obtained from 30 independent runs for each model configuration. These visualizations capture the variability in performance due to different hyperparameter settings and random initialization, providing insight into each model's stability and reliability. The best-performing hyperparameter combinations (those that consistently yielded the highest F1-scores) are summarized in Table 13 in the Appendix. This comprehensive analysis helps ensure that our model selection is both robust and statistically sound.

Our experiments reveal that larger embedding dimensions necessitate deeper neural networks to effectively capture complex relationships in high-dimensional spaces. Specifically, as the embedding size increases from 1536 to 2560, we observe a transition from one to two hidden layers.

6.5 Inference

During inference, the neural network processes each CVE description from the unbalanced dataset to predict its corresponding CWE, as summarized in Table 2. The classification outcomes and performance analysis are discussed in the following section.

7 Model selection results

To evaluate the effectiveness of our proposed approach, we conduct a series of experiments aimed at testing the hypotheses formulated during the methodology. Specifically, we assess the impact of fine-tuning on model performance, compare different neural network architectures, and examine the role of NLP in enhancing prediction accuracy.

Throughout the paper, all experiments are conducted under consistent settings. For each configuration, we train three independent instances and obtain the final prediction by averaging their class-probability outputs, which reduces variance due to random initialization and yields more stable results. As summarized in Table 2, the fine-tuned models outperformed their base counterparts across all metrics. Notably, incorporating the NLP pipeline did not yield consistent improvements, suggesting that relying directly on the original CVE descriptions is more effective.

The enhanced performance of fine-tuned models appears to correlate with the number of parameters; BERT Large, having the most parameters among the classification models, benefits the most from fine-tuning. Among the LLMs, fine-tuning led to substantial improvements: Phi's F1-score increased by over 4.5 percentage points, and LLaMA's by more than 8.5 points. After fine-tuning,

Table 2: Ablation Study: Performance of CVE-to-CWE Mapping Models (Weighted Averages). The table compares baseline models, models with added NLP preprocessing, and fine-tuned models across several architectures.

Model	Weights available	Precision	Recall	F1-score	Accuracy
BERT	✓	68.46%	59.87%	60.69%	59.87%
BERT with NLP	✓	66.56%	58.73%	59.32%	58.73%
BERT fine-tuned	✓	67.98%	60.19%	61.38%	60.19%
BERT Large	✓	68.99%	59.96%	60.85%	59.96%
BERT Large with NLP	✓	67.41%	59.01%	59.39%	59.01%
BERT Large fine-tuned	✓	69.86%	62.53%	63.36%	62.53%
RoBERTa	✓	67.58%	59.00%	59.77%	59.00%
RoBERTa with NLP	✓	67.20%	59.68%	60.81%	59.68%
RoBERTa fine-tuned	✓	67.78%	59.16%	60.17%	59.16%
Phi	✓	73.47%	65.92%	66.92%	65.92%
Phi with NLP	✓	71.71%	64.64%	65.22%	64.64%
Phi fine-tuned	✓	76.66%	70.92%	71.60%	70.92%
LLaMA	✓	72.42%	64.58%	65.17%	64.58%
LLaMA with NLP	✓	70.87%	63.87%	64.22%	63.87%
LLaMA fine-tuned	✓	**77.84%**	**72.46%**	**73.79%**	**72.46%**
GPT (embeddings)	✗	73.13%	67.36%	68.07%	67.36%
GPT (Zero-shot)	✗	71.96%	61.03%	62.27%	61.03%
GPT fine-tuned	✗	72.09%	67.79%	67.86%	67.79%

both models achieve strong performance, with LLaMA slightly outperforming Phi.

These results underscore the significance of fine-tuning in LLM-based classification tasks. Although the base performance of Phi (3B), LLaMA (7B), and GPT (175B) models is relatively comparable, models that support embedding extraction (namely Phi and LLaMA) show consistent performance gains after fine-tuning. To ensure a fair comparison, we selected model versions released around the same time, thereby aligning their training cutoffs. Specifically, the results presented are based on Phi-2, LLaMA-2, and GPT-3.5 Turbo, which are versions that outperform their successors, as further elaborated in Section 10.4. For additional GPT-specific insights, refer to Section 7.2.

7.1 Attention analysis

To further validate the performance improvements achieved through fine-tuning, we analyzed the embeddings and attention vectors associated with the last token, which is the token that aligns with the predicted CWE label. By extracting the tokens and their corresponding attention weights, we can examine how the fine-tuned models adjust their focus to enhance CWE prediction.

Table 3 illustrates the differences in attention patterns between the fine-tuned and non-fine-tuned LLaMA models for CWE-79, which corresponds to Cross-Site Scripting (XSS). Notably, the acronym "XSS" is tokenized as separate tokens: "X" and "SS". While the token "X" is common in natural language, the table highlights the token "SS", which plays a more distinctive role in identifying Cross-Site Scripting. The results demonstrate that its relative attention weight significantly increases in the fine-tuned model compared to the base model, indicating improved focus on relevant security-specific patterns.

The token *'SS'* shifts from position *14* in the non-fine-tuned model to position *2* in the fine-tuned model, reflecting a marked increase in its relative importance for identifying the corresponding CWE.

Table 3: Comparison of CVE Description Relevance Scores Before and After Fine-Tuning. 'FT' denotes Fine-Tuning, while 'no FT' denotes without Fine-Tuning.

Aspect	Details
CVE	"ChurchCRM 4.5.3 and below was discovered to contain a stored cross-site scripting (XSS) vulnerability at /api/public/register/family."
CWE	79 Cross-site scripting (XSS)
Relevance Scores with no FT	0.023814, 0.00923, 0.00894, 0.00639, 0.00624, 0.00615, 0.00607, 0.00605, 0.00604, 0.00602, 0.00579, 0.00575, 0.00575, **0.00554**, 0.00550, …
Tokens with no FT	ability, _vulner, . , _Church, _, _/, register, _below, _at, api, CR, family, X, M, **SS**, . , 5, _(, _stored, -, 4, _script, 3, /, _cross, ing, _a, public, /, _to, _was, _discovered, /, _contain, _and, site,), .
Relevance Scores with FT	0.01889, **0.01060**, 0.00967, 0.00862, 0.00810, 0.00805, 0.00801, 0.00759, 0.00757, 0.00739, …
Tokens with FT	ability, **SS**, . , _vulner, M, register, _below, CR, 4, _Church, 5, _was, 3, _a, _/, _, family, api, _to, . , _contain, _at, _discovered, /, _stored, _(, X, public, _and, . , _cross, -, /, /, ing,), _script, site

7.2 GPT methods comparison

Although GPT underperforms in the overall LLM comparison (Table 2), it is important to understand why a state-of-the-art closed-source model fails to reach state-of-the-art performance in our setting.

As shown in Table 4, GPT embeddings outperform direct GPT-based classification, even when the model is fine-tuned. Since OpenAI does not support extracting embeddings from fine-tuned models, the embedding-based approach necessarily relies on the base GPT model. The key takeaway is that closed-source models, such as GPT (which do not permit embedding extraction from fine-tuned instances), underperform compared to open-weight LLMs that support this capability, highlighting a limitation for downstream tasks such as CVE-to-CWE mapping.

Table 4 compares several approaches. The models are evaluated using macro-average performance metrics, and the last two approaches rely on GPT API calls, effectively representing zero-shot classification methods.

- GPT (embeddings): This method achieves the highest performance, with an F1-score of 58.81%. It involves extracting sentence embeddings from GPT and using them as input for a separate classifier. These embeddings capture rich semantic representations of the CVE descriptions, enabling more effective CWE mapping than direct predictions.

- GPT base: Directly using GPT to predict CWEs results in poor performance, indicating that the base model is not well-suited for this task without further modification.
- GPT fine-tuned: Even when GPT is fine-tuned on the specific task of mapping CVEs to CWEs, the model's performance is still subpar. Fine-tuning improves over the baseline, but it remains far behind the performance achieved by using GPT embeddings.

For fairness, we selected a GPT version consistent in release date with the open-source models used in this study. A broader discussion of model versions and performance comparisons is provided in Section 10.4.

Table 4: Macro-average performance comparison of GPT-based CWE classification. Results are reported for two primary approaches: (1) embedding-based classification (available only for the base model), and (2) direct prediction via the GPT API (both base and fine-tuned). Post-processing refers to mapping GPT outputs only to the 57 CWE candidates.

Approach	Model	Post-processing	Precision	Recall	F1-score
Embedding-based	GPT base	✗	55.99%	64.98%	58.81%
Direct prediction	GPT base	✗	10.74%	8.76%	8.72%
	GPT fine-tuned	✗	28.30%	32.86%	29.88%
	GPT base	✓	54.27%	44.25%	44.05%
	GPT fine-tuned	✓	52.63%	61.10%	55.57%

To ensure the comparison focuses solely on the 57 selected classification classes, we post-processed the outputs from both the GPT baseline and the fine-tuned model, filtering out any CVEs that were mapped to CWEs outside of this set. Despite this filtering step, using embeddings for classification proved more effective than directly relying on CWE predictions from the fine-tuned model.

The most effective approach for the CWE mapping task is to first extract embeddings from GPT and then train a separate neural network on these embeddings. This embedding-based method consistently outperforms direct classification using the fine-tuned GPT models.

8 2-step classification

Building on the performance assessments of individual models, we further enhance classification by leveraging the hierarchical structure of CWEs. The NIST CWE taxonomy organizes weaknesses into two levels: broad parent categories (e.g., Buffer Overflow) and more specific child variants (e.g., Buffer Overflow Write, Buffer Overflow Read).

To leverage this structure, we adopt a two-step classification strategy. First, a top-level classifier predicts the parent CWE. Then, conditioned on this prediction, a specialized classifier (trained only on that parent and its children) selects the most appropriate subtype. This design restricts the second-stage prediction to a small, semantically coherent subset of CWEs that are highly similar to one another, thereby improving fine-grained accuracy. This stands in contrast to other approaches, such as [24], where the distinction between parent and child CWEs provides little practical benefit. In their design, the child classifier still selects from the full pool of all possible CWE children, meaning the hierarchical split does not narrow the search space nor exploit semantic proximity.

We selected the best-performing model from our previous evaluations (fine-tuned LLaMA) as the CVE embedder. Its embeddings feed into a two-step neural network that drives the hierarchical classification. The overall process is illustrated in Figure 4.

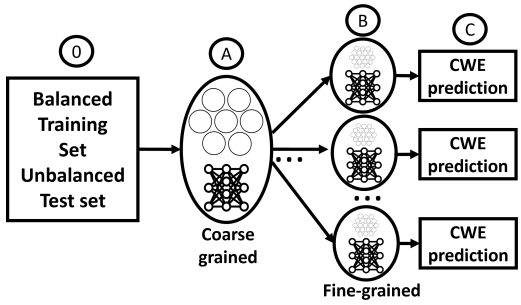


Figure 4: CWE prediction using the coarse-fine-grained predictions

8.1 Coarse-grained

In the coarse-grained stage, we leverage the hierarchical structure defined by NIST, where each CWE is associated with a broader CWE class, either matching the NIST classification or corresponding to its parent category. In this first stage, a classifier is trained and tested on these high-level CWE classes, focusing on distinguishing primary differences between them. For example, the classifier differentiates between CWE-74 (Injection) and CWE-119 (Buffer Overflow). This initial categorization groups CVE descriptions into broad, meaningful clusters, laying the groundwork for subsequent fine-grained analysis.

8.2 Fine-grained

Following coarse-grained classification, a specialized neural network is deployed for fine-grained classification within each predicted CWE class. This network is designed to discern not only the specific CWE base types and their variants but also to recognize when the most accurate classification corresponds to the parent CWE rather than a more detailed child CWE. For example, consider the vulnerability CVE-2009-2824:

> Multiple buffer overflows in Apple Type Services (ATS) in Apple Mac OS X 10.5.8 allow remote attackers to execute arbitrary code via a crafted embedded font in a document.

Although this description provides sufficient detail to classify the vulnerability as CWE-119 (Improper Restriction of Operations within the Bounds of a Memory Buffer), it does not offer enough information to determine whether it specifically involves a buffer overflow read or write. In such cases, assigning the parent CWE (CWE-119) is more appropriate.

This type of fine-grained classification is particularly challenging and remains largely unaddressed in prior research. Existing studies, such as [2], typically classify CWEs to the lowest level possible. This approach is easier because all child CWEs within a class share many common characteristics with the parent CWE, whereas the child CWEs themselves often exhibit differentiating features.

Other approaches, like [13], focus solely on predicting whether a link exists between two CWEs in the hierarchy, considering the prediction successful if the correct CWE is connected by one edge in the hierarchy.

8.3 CWE Prediction

Finally, we evaluate our hierarchical classification framework using the dataset described in Section 5.1. Our approach integrates a single model for coarse-grained prediction with specialized models for fine-grained classification, one for each CWE parent (a total of fourteen), to collectively determine the final CWE prediction for each CVE description.

9 Results

In this section, we compare the results obtained from our two-step classification framework. Table 5 summarizes the performance of our approach (highlighted in bold). Across the 57 CWE classes, our method achieves a precision exceeding 78% and a recall approaching 75%. Notably, there is no clear correlation between the number of samples per class and the final F1 score. For instance, CWE-79, with 3,000 training samples, exhibits the highest precision at 99.80%, whereas CWE-611, despite having less than one-third of the samples of CWE-79, achieves the best recall.

The overall performance drop can be primarily attributed to the first classification step, which registers an F1-score below 84%. This result suggests that the fine-grained classification stage contributes significantly to performance gains. A contributing factor is that the coarse-grained classifier distinguishes among 28 broader classes, whereas the fine-grained classifiers handle a maximum of only 6 classes within each group, thereby benefiting from a more focused and less complex prediction task.

9.1 Comparison

In Section 4, we reviewed several approaches for automatically mapping CVEs to CWEs. Notably, [38] recently proposed an unsupervised learning strategy, while [25] and [32] achieved strong results by directly assigning CWE labels from vulnerability descriptions. The former employs a chain-of-thought prompting technique with a large language model, whereas the latter adopts a supervised method based on Text-CNN and BiGRU architectures. In this work, we re-evaluated these three approaches using the same dataset of CVE descriptions and corresponding CWE labels described in Section 5.1, retraining the models where necessary. Our results, summarized in Table 5, demonstrate the superior performance of our proposed architecture.

Furthermore, we attempted to replicate the methods proposed in [13], [2], however, we encountered reproducibility issues (specifically difficulties in executing the tools). In addition, these models were trained on a hierarchical structure that differs from our own. An analogous scenario emerged with [4], where the authors apply the same principles as [2] and report a classification accuracy of less than 70% on 25 CWEs.

Our method achieves an F1-score of 75.07%, representing a significant improvement over the 32.35% achieved by the unsupervised approach of [38] and 23.11% improvement on F1 on the LLM-based chain of thought. Supervised learning tends to outperform unsupervised methods when sufficient labeled data is available, and prior results (Section 7) already suggested that LLM-based prompting performs worse than LLM-embedding-based architectures for this task.

Table 5: Improvement over the state-of-the-art with our two-step approach on weighted average.

Model	Precision	Recall	F1-score	Accuracy
Unsupervised [38]	45.63%	35.41%	32.45%	35.41%
LLM based [25]	69.08%	58.56%	60.96%	58.56%
BiGRU-TextCNN [32]	76.18%	70.61%	71.53%	70.61%
Our approach	**78.57**%	**74.90** %	**75.07**%	**74.90**%

Compared to the best-performing supervised baseline (BiGRU-TextCNN), our method achieves a 5% improvement in F1-score. To isolate the contribution of each architectural enhancement, we first compare BiGRU-TextCNN with the fine-tuned Llama-based model described in Section 7, followed by the inclusion of our hierarchical coarse-to-fine classification strategy. Overall, the 5% improvement from our approach can be attributed to:

- Approximately 3% is attributable to the use of a fine-tuned transformer-based embedding model instead of BiGRU and CNN layers.
- The remaining 2% is due to our hierarchical two-step classification design, which better aligns with the structured nature of the CWE taxonomy.

9.2 Robustness and Statistical significance

In Section 9.1, we reported, for each approach, the performance obtained from a single evaluation. This yields a single result per model and metric, which is affected by randomness and does not explicitly quantify variability across repeated evaluations. We therefore evaluate all methods over 10 independent runs and report the mean and standard deviation of each metric, as presented in Table 6.

Table 6: Performance of different methods (mean ± standard deviation over multiple runs).

Model	Precision	Recall	F1-score	Accuracy
Unsupervised [38]	46.38 ± 2.51	35.16 ± 0.75	32.38 ± 0.16	35.16 ± 0.75
LLM based [25]	68.82 ± 0.28	58.35 ± 0.18	60.65 ± 0.19	58.35 ± 0.18
BiGRU-TextCNN [32]	76.02 ± 0.31	69.84 ± 0.45	70.74 ± 0.42	69.84 ± 0.45
Llama fine-tuned (Sec. 7)	77.24 ± 0.63	70.93 ± 1.75	71.48 ± 1.63	70.93 ± 1.75
Our approach	**78.47 ± 0.30**	**73.01 ± 0.92**	**73.56 ± 0.60**	**73.01 ± 0.92**

Our approach achieves the highest scores across all metrics, consistently outperforming the unsupervised method, LLM-based, the BiGRU-TextCNN model, and the fine-tuned LLaMA. While the fine-tuned LLaMA constitutes the first stage of our method, our approach further incorporates the hierarchical mapping, which yields the observed performance gains.

To address the question of whether these improvements are due to chance, we conducted a statistical significance analysis comparing our method with the BiGRU-TextCNN baseline. Both models

were independently trained and evaluated 30 times. After verifying the normality assumption on the per-run F1-scores using the Shapiro-Wilk test, we applied a paired t-test and obtained a p-value of 2^{-20} that is orders of magnitude smaller than the commonly used significance threshold of 0.05. This confirms that the performance improvement of our proposed approach ('Our approach') over BiGRU-TextCNN is statistically significant.

9.3 Coarse Grained only

In the first classification stage, which distinguishes among 28 CWE parent classes, our approach achieves a weighted average F1-score of 83.80%. Notably, there is no clear correlation between the volume of training data and performance; for example, despite CWE-20 having the largest training sample size, it records the third-lowest F1-score at 44.57%. The overall performance in this step is primarily limited by a low recall rate. Specifically, the model frequently misclassifies instances of CWE-20 ('Improper Input Validation') as follows:

(1) CWE-74 (Injection) 15.68% of the time,
(2) CWE-200 (Exposure of Sensitive Information to an Unauthorized Actor) 9.5% of the time,
(3) CWE-119 (Improper Restriction of Operations within the Bounds of a Memory Buffer) 9.25% of the time.

This coarse-grained classification is not only a prerequisite for the subsequent fine-grained step but can also be valuable on its own for certain applications. For instance, identifying the CWE parent can facilitate vulnerability prioritization based on general categories, enabling faster, broader classifications that support risk assessment and mitigation strategy development. This approach is especially suitable for applications that need to quickly determine whether a vulnerability pertains to broad issues such as *'buffer overflow'* or *'information exposure'*. Further discussion of the coarse-grained classification performance is provided in Appendix J.

9.4 Fine-grained only

For each predicted CWE class from the coarse-grained step, we train a specialized neural network to distinguish among the subclass labels and their corresponding parent CWE. One of the most challenging aspects is differentiating the parent CWE from its children when subtle differences exist. This fine-grained stage is designed to capture these nuances, enabling our system to accurately assign the final CWE label.

The effectiveness of fine-grained classification depends on several factors:

9.4.1 Number of subclasses. Performance varies considerably based on the number of subclasses within a CWE parent. Some classes, like CWE-74, contain as many as six subclasses, whereas others have no subclasses due to the cut-off threshold (discussed in Section 10), which excludes subclasses with insufficient instances.

9.4.2 Semantic Differences Among Subclasses. Subclasses with distinct semantic meanings, such as CWE-787 (out-of-bounds write) versus CWE-125 (out-of-bounds read), are easier to differentiate due to clear operational differences (e.g., write versus read operations). However, when parent and child CWEs share similar characteristics, such as the relationship between CWE-119 (Buffer Overflow parent) and CWE-120 (Classical Buffer Overflow), the distinctions become more subtle, leading to reduced classification performance.

9.4.3 Presence of the CWE parent. The parent class is not always present in the final classification because they don't meet the dataset threshold (Section 5.1). The inclusion or exclusion of the parent label affects the overall performance. To illustrate the different scenarios handled by our fine-grained classifiers, Table 7 presents detailed performance metrics for three representative cases:

- CWE-74 (multiple subclasses and parent CWE): This class is misclassified as CWE-79 in 63% of predictions and is correctly classified only 27% of the time, significantly reducing its precision.
- CWE-732 (one subclass and parent CWE): Despite having only one subclass, this class shows poor F1 scores due to close ties between the parent CWE-732 (Incorrect Permission Assignment for Critical Resource) and its child CWE-276 (Incorrect Default Permissions).
- CWE-672 (Subclass-only Representation): Some classes are represented only by their subclasses in the final classification, as seen with CWE-672, which is excluded due to fewer than 315 instances.

Table 7: Classification reports that showcase the three different scenarios. The first and the second show the impact of having the CWE parent in the final classification and are differentiated by the number of subclasses. The third highlights the absence of the CWE parent in the final classification and the improvement in the performances.

CWE parent	CWE children	Precision	Recall	F1-score	Accuracy
CWE-74		27.08%	54.93%	36.28%	54.93%
	CWE-77	83.33%	47.30%	60.34%	47.30%
	CWE-78	72.70%	90.88%	80.78%	90.88%
	CWE-79	99.95%	88.08%	93.64%	88.08%
	CWE-89	99.78%	96.66%	98.20%	96.66%
	CWE-94	53.25%	92.47%	67.58%	92.47%
weighted avg		92.61%	88.56%	89.65%	88.56%
CWE-732		73.81%	63.27%	68.13%	63.27%
	CWE-276	53.85%	65.62%	59.15%	65.62%
weighted avg		65.92%	64.20%	64.59%	64.20%
CWE-672					
	CWE-415	95.12%	92.86%	93.98%	92.86%
	CWE-416	99.08%	99.38%	99.23%	99.38%
weighted avg		98.63%	98.64%	98.63%	98.64%

Overall, the fine-grained classification step is designed to resolve the subtle distinctions among subclasses and improve the granularity of our CWE predictions. The performance of each fine-grained classifier varies according to the number of subclasses, the semantic differences between them, and whether the parent class is included in the final prediction.

10 Discussion

This section analyzes key challenges in our approach and discusses several strategies aimed at enhancing the performance of our two-step neural network framework. We focus on four main areas:

addressing data imbalance, evaluating the impact of cut-off thresholds, assessing potential data leakage in the base model training set, and comparing different LLM model families and versions.

10.1 Data imbalance

Initial attempts to mitigate data imbalance through oversampling and class weighting for underrepresented classes proved ineffective. As emphasized by Wheelus et al. [43], there is no universally optimal solution because optimal techniques depend on the dataset's specific characteristics. Training machine learning models on imbalanced data often leads to bias toward dominant classes [8].

To address the imbalance, we experimented with methods such as SMOTE [9], which generates synthetic samples for minority classes, and class weighting, which assigns higher importance to underrepresented classes during training. However, these approaches resulted in a 2–5% drop in performance.

As described in Section 10.2, we adopted a cut-off threshold designed to retain over 95% of CVEs associated with a CWE, based on the frequency distribution in the Known Exploited Vulnerabilities (KEV) database. We also considered alternative strategies, including accounting for the temporal evolution of CWE distributions.

Table 10 in the Appendix demonstrates that while earlier years were dominated by 'buffer overflow' vulnerabilities, recent data indicate a higher prevalence of 'injection' vulnerabilities. To further understand this evolution, we analyzed data from 2022 to 2023 (data CVEs for 2024 and 2025 are still under revision and update from NIST experts). As illustrated in Figure 15 in the Appendix, there is no distinct inflection point where the frequency of CWEs in 2023 suddenly exceeds that of the previous year. Consequently, we decided not to adopt a threshold based on temporal trends, as the absence of a clear shift would render the cut-off subjective rather than strictly data-driven.

10.2 Cut-off threshold justification and limitations

In our supervised approach, it is essential to define a cut-off threshold, since, among the 130 CWEs, some have few associated samples. For example, CWE-920 (Improper Restriction of Power Consumption) is linked to only 3 CVEs. We determined our threshold by analyzing the KEV database. Specifically, we based the threshold on the number of CVEs associated with the least frequent CWE among the top 30 most exploited base CWEs. This analysis led us to exclude any CWE with fewer than 315 associated CVEs, a threshold that covers 95% of all CVEs linked to a CWE. Consequently, our focus is on the 57 most common CWEs.

To assess how this threshold impacts classification performance, we systematically evaluated our two-step neural network approach across various thresholds. Specifically, we explored thresholds slightly below and above our primary choice (resulting in 52 and 62 classes, respectively), thresholds substantially below and above (26 and 91 classes), and finally, a scenario where no threshold was applied, including all available CWEs. This comprehensive evaluation enabled us to rigorously compare the performance of our two-step method against a simpler one-step approach, determining both the consistency of observed improvements across varying class selections and quantifying their magnitude (see Table 16 in the Appendix).

Figure 5 compares the F1-scores of the one-step and two-step neural network models across different class counts. The results clearly demonstrate that the two-step neural network consistently outperforms the one-step model, with the performance gap widening as the number of classes increases. This trend suggests that the two-step architecture is more effective at managing the complexity of multi-class scenarios, yielding enhanced performance, particularly when handling larger class sets.

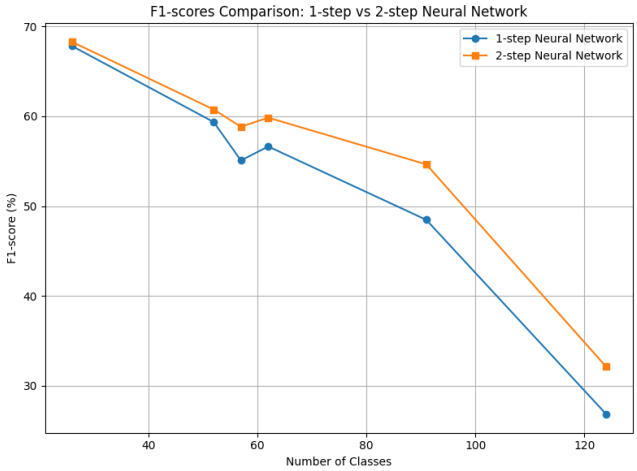


Figure 5: F1-score Comparison of 1-step and 2-step Neural Network Models Across Different Class Counts

10.3 Potential data leakage in the base model training set

A key concern when evaluating base models (e.g., LLaMA) and their fine-tuned derivatives is the risk of data leakage. Since these models are often trained on publicly available datasets, portions of the CVE dataset may have been included in their training data. Such overlap can inflate performance metrics if the test set inadvertently contains examples that were seen during pretraining.

One strategy to mitigate this issue is to evaluate the models exclusively on data published after the release of the base model, ensuring no overlap with the training dataset. To address this, we ran the fine-tuned LLaMA model in inference mode on CVEs published after the model's release. This evaluation was restricted to CVEs with associated CWEs in the NIST database, according to the CWE-1003 guidelines, resulting in a new test set of 2,777 CVEs.

Additionally, to determine whether performance changes are due to the base models or the inherent complexity of the new test set, we evaluated alternative approaches that operate independently of the base model. Table 8 compares macro-average metrics for models on the original test set (12,845 samples) versus the new post-release test set. We focused on the macro-average metric because the limited timeframe of post-release data introduces potential biases; for example, CWE-476 ('NULL Pointer Dereference') accounts for 15% of the new test set but only 1.7% of the original multi-year test set. As shown in Table 8, all models exhibited a consistent performance drop of approximately 11 percentage points on the new test set,

regardless of the model architecture. This decline indicates that the reduced performance is primarily due to the models' challenges in accurately predicting CWEs for this specific test set, rather than being a result of data leakage. If data leakage were a significant factor, we would expect a more pronounced performance drop compared to other models. For additional insight, we also included performance metrics based on the weighted average in Table 15 in the Appendix.

Table 8: Comparison of macro-average metrics across models on two distinct test sets: the Original test set (12,845 samples) and a New test set (2,777 samples) explicitly curated to exclude data from the Llama base model training set.

	Original test set (12845 supports)			New test set (2777 supports)		
Model	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
TextCNN	60.81%	68.42%	62.64%	53.69%	54.74%	49.53%
BiGRU-TextCNN	61.26%	68.58%	62.77%	54.50%	57.64%	52.25%
Our approach	64.60%	68.24%	**64.83**%	59.63%	56.02%	**53.81**%

10.4 Model Versions and Related Performance

To better understand the implications of model architecture and versioning on the task of CVE-to-CWE mapping, we compared two families of models, LlaMA and GPT, and analyzed the impact of fine-tuning on each.

While both model families support fine-tuning, LLaMA offers the additional advantage of retrieving the embeddings for both versions (fine-tuned and non-fine-tuned), which is essential for downstream integration. In contrast, GPT models do not currently support embedding extraction for custom fine-tuned versions. As a result, this section focuses on understanding the effect of version upgrades and fine-tuning within each model family, rather than comparing them directly (as already discussed in Section 7).

As expected, newer base models generally outperform their predecessors across all key metrics, reaffirming the continuous improvement in large language model capabilities over time.

However, the effects of fine-tuning differ by model generation. In terms of macro-averaged performance, which gives equal weight to all classes, fine-tuning consistently improves results across all models and versions. This suggests that fine-tuning enhances the model's ability to generalize to underrepresented classes.

In contrast, weighted average performance, which emphasizes accuracy on more frequent classes, tells a different story. Older models benefit substantially from fine-tuning, with performance improvements of +10.53% for LLaMA2 and +8.97% for GPT-3.5 in weighted F1-score. For newer models, however, fine-tuning results in performance degradation: negligible in the case of LLaMA3, but more pronounced for GPT-4.1, which shows a 7.96% decline in weighted F1-score. This divergence points to a trade-off: while fine-tuning improves precision and recall for rare CWE classes, it can degrade performance on frequent classes, particularly those most common in the test set. In our data, the top CWE classes, capped at 3,000 training examples yet comprising roughly two-thirds of the test set, are disproportionately affected. After fine-tuning, both LLaMA3 and GPT-4.1 underperform their base versions on these frequent classes, with weighted F1-score declines of 2.23% and 6.68%, respectively.

Based on these findings, the model selected for the final pipeline is the fine-tuned LLaMA2, which demonstrated the best overall performance after fine-tuning. While newer models generally achieve better performance, fine-tuning does not always lead to further improvements, as one might expect. This observation highlights an important area for future research: understanding the effects of fine-tuning in scenarios with class imbalance, especially when base models already perform well on dominant classes.

Table 9: Performance comparison across model versions

Model	Fine-tune	Prec	Rec	F1	Acc
LLaMA2	✗	72.42%	64.58%	65.17%	64.58%
	✓	**77.91**%	**70.89**%	**72.03**%	**70.89**%
LLaMA3	✗	76.47%	70.21%	70.48%	70.21%
	✓	76.71%	69.44%	70.13%	69.44%
GPT-3.5	✗	71.96%	61.03%	62.27%	61.03%
	✓	72.09%	67.79%	67.86%	67.79%
GPT-4.1	✗	76.37%	64.41%	67.73%	64.41%
	✓	70.68%	62.37%	62.34%	62.37%

11 Conclusions and future work

This research demonstrates the viability and efficacy of leveraging language models, particularly LLMs, for mapping CVE descriptions to their corresponding CWEs. First, we fine-tuned various language models specifically for the CVE-to-CWE mapping task and identified the best-performing model. Building on this, we developed a two-step neural network that further enhanced performance by leveraging the hierarchical structure of the CWE taxonomy. This combined methodology delivered a 5% improvement in F1-score over previous supervised state-of-the-art techniques.

To the best of our knowledge, no prior work has integrated LLM-based representations with hierarchical CWE family classification in this manner, making our approach both methodologically relevant and domain-specific.

Looking ahead, a promising direction to address the persistent data imbalance issue is to generate additional synthetic data for underrepresented CWEs. Future work could explore advanced data augmentation techniques or utilize generative models, such as generative adversarial networks (GANs) [17], to create realistic and diverse CVE descriptions. This approach could enhance both the balance and diversity of data, potentially further improving mapping accuracy.

Another valuable extension of this work is the application of our hierarchical classification approach to other CTI taxonomies. Domains such as TTPs and CAPEC share similar hierarchical structures, suggesting that coarse-to-fine prediction strategies could yield comparable performance gains. Exploring these adjacent taxonomies would further validate the generalizability of our methodology.

References

[1] Ehsan Aghaei, Xi Niu, Waseem Shadid, and Ehab Al-Shaer. 2022. Securebert: A domain-specific language model for cybersecurity. In *International Conference on Security and Privacy in Communication Systems*. Springer, 39–56.

[2] Ehsan Aghaei, Waseem Shadid, and Ehab Al-Shaer. 2020. Threatzoom: Hierarchical neural network for cves to cwes classification. In *International Conference on Security and Privacy in Communication Systems*. Springer, 23–41.

[3] Ahmed Agiza, Marina Neseem, and Sherief Reda. 2024. MTLoRA: Low-Rank Adaptation Approach for Efficient Multi-Task Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16196–16205.

[4] Massimiliano Albanese, Olutola Adebiyi, and Frank Onovae. [n. d.]. CVE2CWE: Automated Mapping of Software Vulnerabilities to Weaknesses Based on CVE Descriptions. ([n. d.]).

[5] Anonymous. 2024. Mapping CVE to CWE by leveraging CWE Hierarchy with LLMs. Accessed on November 13, 2024, from https://anonymous.4open.science/r/2-layer-nn-C257/README.md.

[6] Masaki Aota, Hideaki Kanehara, Masaki Kubo, Noboru Murata, Bo Sun, and Takeshi Takahashi. 2020. Automation of Vulnerability Classification from its Description using Machine Learning. In *2020 IEEE Symposium on Computers and Communications (ISCC)*. 1–7. doi:10.1109/ISCC50000.2020.9219568

[7] Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. 2025. Small Language Models are the Future of Agentic AI. *arXiv preprint arXiv:2506.02153* (2025).

[8] Rok Blagus and Lara Lusa. 2013. SMOTE for high-dimensional class-imbalanced data. *BMC bioinformatics* 14 (2013), 1–16.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. 2002. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research* 16 (June 2002), 321–357. doi:10.1613/jair.953

[10] Common Vulnerabilities and Exposures. 2024. *Common Vulnerabilities and Exposures*. https://www.cve.org/

[11] Andrei Costin, Hannu Turtiainen, Narges Yousefnezhad, Vadim Bogulean, and Timo Hämäläinen. 2024. Evaluating Zero-Shot Chatgpt Performance on Predicting CVE Data From Vulnerability Descriptions. In *European Conference on Cyber Warfare and Security*, Vol. 23. 576–584.

[12] Siddhartha Shankar Das, Edoardo Serra, Mahantesh Halappanavar, Alex Pothen, and Ehab Al-Shaer. 2021. V2W-BERT: A Framework for Effective Hierarchical Multiclass Classification of Software Vulnerabilities. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*. 1–12. doi:10.1109/DSAA53316.2021.9564227

[13] Siddhartha Shankar Das, Edoardo Serra, Mahantesh Halappanavar, Alex Pothen, and Ehab Al-Shaer. 2021. V2w-bert: A framework for effective hierarchical multiclass classification of software vulnerabilities. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 1–12.

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

[15] Gemini Team et al. 2024. Gemini: A Family of Highly Capable Multimodal Models. arXiv:2312.11805 [cs.CL] https://arxiv.org/abs/2312.11805

[16] Hugo Touvron et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[17] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. arXiv:1406.2661 [stat.ML] https://arxiv.org/abs/1406.2661

[18] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. 2023. Textbooks are all you need. *arXiv preprint arXiv:2306.11644* (2023).

[19] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, et al. 2021. Pre-trained models: Past, present and future. *AI Open* 2 (2021), 225–250.

[20] Zhuobing Han, Xiaohong Li, Hongtao Liu, Zhenchang Xing, and Zhiyong Feng. 2018. Deepweak: Reasoning common software weaknesses via knowledge graph embedding. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 456–466.

[21] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).

[22] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *Comput. Surveys* 55, 12 (2023), 1–38.

[23] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL] https://arxiv.org/abs/2310.06825

[24] Kethan Kota, A Manjunatha, et al. 2024. CWE prediction using CVE description-the semantic similarity approach. *Procedia Computer Science* 235 (2024), 1167–1178.

[25] Xin Liu, Yuan Tan, Zhenghang Xiao, Jianwei Zhuge, and Rui Zhou. 2023. Not the end of story: An evaluation of ChatGPT-driven vulnerability description mappings. In *Findings of the Association for Computational Linguistics: ACL 2023*. 3724–3731.

[26] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[27] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 9802–9822. doi:10.18653/v1/2023.acl-long.546

[28] Microsoft Research Blog. [n. d.]. Phi-2: The Surprising Power of Small Language Models. https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/. Accessed:21-3-2024.

[29] National Institute of Standards and Technology. [n. d.]. *National Vulnerability Database*. National Institute of Standards and Technology. https://nvd.nist.gov/

[30] National Vulnerability Database (NVD). 2024. *NVD Data Feeds*. https://nvd.nist.gov/vuln/data-feeds

[31] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.

[32] Mengyuan Pan, Po Wu, Yiwei Zou, Chong Ruan, and Tao Zhang. 2023. An automatic vulnerability classification framework based on BiGRU-TextCNN. *Procedia Computer Science* 222 (2023), 377–386.

[33] Shengyi Pan, Lingfeng Bao, Xin Xia, David Lo, and Shanping Li. 2023. Fine-grained commit-level vulnerability type prediction by CWE tree structure. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 957–969.

[34] Nicola Piovesan, Antonio De Domenico, and Fadhel Ayed. 2024. Telecom Language Models: Must They Be Large? *arXiv preprint arXiv:2403.04666* (2024).

[35] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[36] Shabana Rehman and Khurram Mustafa. 2012. Software design level vulnerability classification model. *International Journal of Computer Science and Security (IJCSS)* 6, 4 (2012), 238.

[37] Stefano Simonetto and Peter Bosch. 2023. Are we reasoning about cloud application vulnerabilities in the right way?. In *8th IEEE European Symposium on Security and Privacy*.

[38] Stefano Simonetto, Thijs Sebastiaan van Ede, Peter Bosch, and Willem Jonker. 2024. Text2Weak: mapping CVEs to CWEs using description embeddings analysis. In *4th Workshop on Artificial Intelligence-Enabled Cybersecurity Analytics*.

[39] Heydar Soudani, Evangelos Kanoulas, and Faegheh Hasibi. 2024. Fine Tuning vs. Retrieval Augmented Generation for Less Popular Knowledge. *arXiv preprint arXiv:2403.01432* (2024).

[40] The MITRE Corporation. [n. d.]. Common Weakness Enumeration (CWE). https://cwe.mitre.org/. Accessed on April 6, 2026.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[42] Qian Wang, Yuying Gao, Jiadong Ren, and Bing Zhang. 2023. An automatic classification algorithm for software vulnerability based on weighted word vector and fusion neural network. *Computers & Security* 126 (2023), 103070.

[43] Charles Wheelus, Elias Bou-Harb, and Xingquan Zhu. 2018. Tackling class imbalance in cyber security datasets. In *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. IEEE, 229–232.

A CVE Trends

Figure 6 shows the trends of increased number of reported CVEs and the lack of assigned CWE labels.

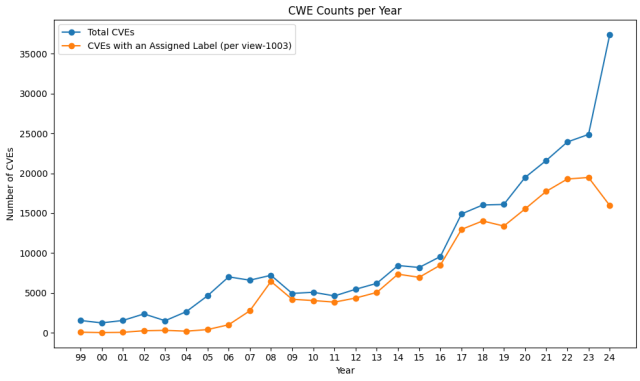


Figure 6: The graph displays two trends: the total number of CVEs reported each year and the subset of CVEs that have an assigned a CWE label according to the 1003-view.

B Distribution of CVEs across CWE labels

Figure 7 shows the distribution of CVEs across CWE labels

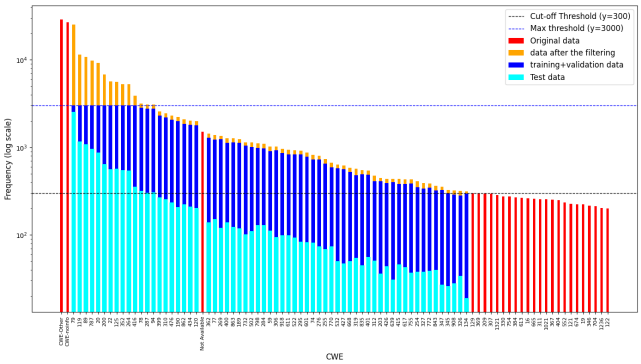


Figure 7: Distribution of CVEs across CWE labels (note: y-axis is in logarithmic scale)

C Cut-out CWEs due to insufficient samples

The following classes and their respective subcategories are being excluded due to their insufficient representation, as they do not reach the cut-off threshold:

- CWE-116: Improper Encoding or Escaping of Output
- CWE-330: Use of Insufficiently Random Values
- CWE-407: Inefficient Algorithmic Complexity
- CWE-436: Interpretation Conflict
- CWE-662: Improper Synchronization
- CWE-674: Uncontrolled Recursion
- CWE-697: Incorrect Comparison
- CWE-922: Insecure Storage of Sensitive Information

D Top 3 CWE IDs per Year

This section summarizes the yearly distribution of the three most frequent CWE identifiers, reporting their absolute CVE counts and relative proportions. The table provides a longitudinal view of shifts in dominant vulnerability classes, enabling the identification of patterns and emerging trends across the CVE dataset.

Table 10: Top 3 CWE IDs per Year with Corresponding CVE Counts and Percentage Shares. Note: NA stands for 'Not available'

Year	Total CVE	1st CWE	2nd CWE	3rd CWE
99	73	264 13.70%	200 13.70%	20 12.33%
00	32	200 15.62%	120 12.50%	264 9.38%
01	58	119 17.24%	22 10.34%	20 8.62%
02	258	119 16.67%	79 13.18%	20 12.40%
03	303	119 19.14%	79 14.19%	20 10.56%
04	213	119 13.62%	264 12.21%	79 11.74%
05	420	119 16.67%	89 12.38%	79 10.48%
06	999	94 20.52%	119 15.12%	79 12.61%
07	2665	119 18.16%	79 14.18%	94 11.37%
08	6399	89 23.14%	79 15.39%	119 9.05%
09	4192	79 17.89%	89 16.08%	119 13.29%
10	4028	79 15.14%	89 14.60%	119 13.06%
11	3834	119 16.74%	79 13.72%	20 11.06%
12	4343	79 19.41%	119 15.17%	264 13.22%
13	5037	79 15.86%	119 15.01%	264 12.19%
14	7324	310 21.07%	79 14.15%	119 10.98%
15	6845	119 15.98%	79 14.29%	200 10.97%
16	8204	119 15.37%	200 11.69%	79 10.07%
17	12715	119 15.73%	79 11.99%	200 9.50%
18	13742	79 15.37%	20 7.46%	787 6.75%
19	12799	79 14.13%	787 10.84%	125 6.37%
20	14882	79 15.59%	787 9.80%	125 5.01%
21	17458	79 15.41%	787 8.94%	125 4.35%
22	19154	79 15.14%	787 10.54%	89 8.49%
23	19264	79 20.99%	89 8.71%	787 8.49%
24	38459	NA 30.80%	79 13.50%	89 5.97%
25	19335	NA 56.07%	89 7.40%	74 6.31%

E NLP pipeline

CVE-2010-1459

The NeXTDecode function in tif_next.c in LibTIFF allows remote attackers to cause a denial of service (out-of-bounds write) via a crafted TIFF image, as demonstrated by libtiff5.tif.

nextdecod function tif_next.c libtiff allow remot attack caus denial servic (out-of-bound write) via craft tiff imag , demonstr libtiff5.tif .

Figure 8: Description of CVE-2015-8784 before and after the NLP pipeline. In different colors are highlighted the different NLP preprocessing parts except for the tokenization which is done and is not visible in the end result: lower-case, stop-word removal, lemmatization, stemming .

F LoRA

LoRA provides controllable trade-offs between fine-tuning efficiency and downstream performance [3]. The specific transformer

modules targeted for LoRA fine-tuning are illustrated in Figure 9 and described as follows:

- **QKV Computation in Attention Layers:** The Query, Key, Value (QKV) computation, integral to the attention mechanism, is a prime candidate for low-rank adaptation. Fine-tuning this component enables task-specific modifications to the attention mechanism, enhancing the model's ability to process and interpret task-relevant visual inputs.
- **Projection Layer:** The projection layer maps the output of the attention mechanism back to the original feature space. By fine-tuning this layer, we ensure that the projected features are more aligned with the downstream task, improving task-specific performance.
- **Feedforward Layers in the MLP Block:** These layers, consisting of two fully connected layers (FC1 and FC2) with a nonlinear activation function in between, transform the attention output into the final feature representation. Fine-tuning these layers optimizes the model's ability to generate feature representations tailored to specific tasks, facilitating improved performance in subsequent stages or task-specific decoders.

The primary variation across models is in their size, as larger models involve training a higher number of parameters. Notably, LoRA's fixed values for α and r are used across models: α controls the scaling factor for the low-rank update matrices, affecting the magnitude of the adjustments, while r determines the rank of the decomposition, thereby setting the capacity of the added trainable matrices.

Fine-tuning is fundamentally aimed at decreasing the loss. Our experiments demonstrated that higher values of α and r lead to better training outcomes. We experimented for each model with different combinations of r and α, with detailed results presented in Appendix F.1. Our results indicate that setting $r = 64$ and $\alpha = 128$ minimizes validation loss most effectively across all models, as shown in Figure 3. In this picture are shown the differences across the 6 epochs in the fine-tuning for all the models except for GPT, which has been fine-tuned following the instructions given by OpenAI, which suggested using only one epoch as shown in Figure 12a in Sec F.1.

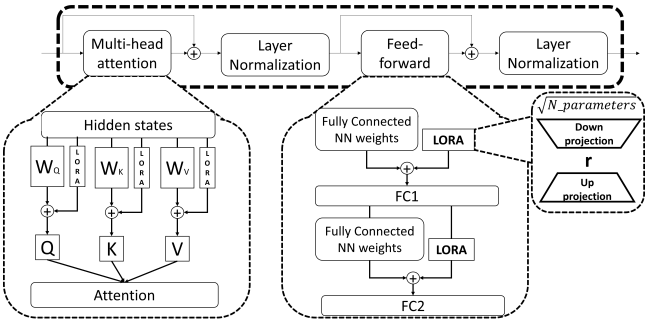


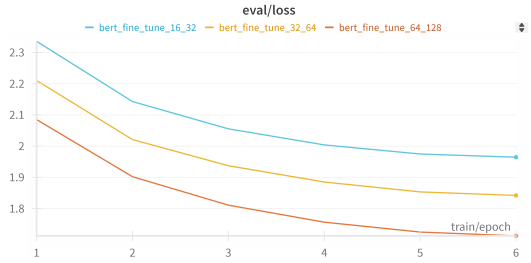
Figure 9: Lora fine tuning on transformers modules

F.1 Fine-tuning results

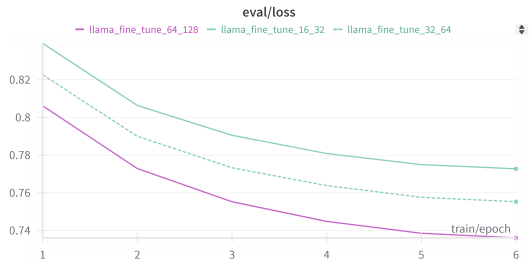
Here we discuss how we fine-tuned the different models using LoRA. Table 12 provides an overview of the used models and their corresponding LoRa configurations. We plot the corresponding validation losses per training epoch for these configurations for BERT, LLaMA and GPT in Figure 12.

Table 11: LoRA Configuration: as suggested by the LoRa original paper, we always adopted the ratio 2 between r and α

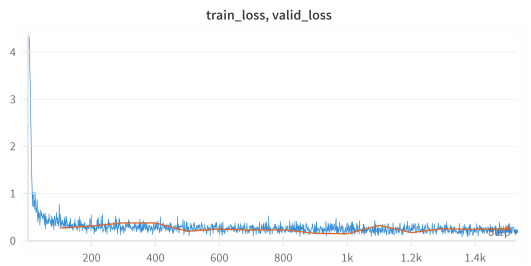
Parameter	Value
r	16,32,64
α	32,64,128
target modules	q_proj,k_proj, v_proj, dense, fc1, fc2
lora dropout	0.05
task type	CAUSAL_LM



(a) BERT



(a) LLaMA



(a) GPT

Figure 12: Fine-tuning evaluation performance of selected transformer-based models.

Table 12: Model parameters and LoRA configurations.

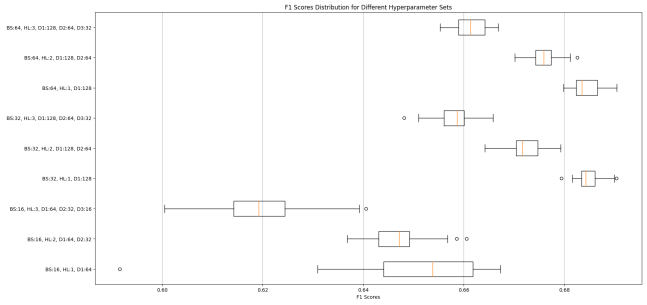
Model	n of param	trainable param	%	r	α
LLaMA3	8,068,009,9848	37,748,736	0.47%	64	128
LLaMA2	6,750,998,528	12,582,912	0.19%	16	32
	6,763,581,440	25,165,824	0.37%	32	64
	6,788,747,264	50,331,648	0.74%	64	128
Phi	2,803,276,800	23,592,960	0.84%	16	32
	2,826,869,760	47,185,920	1.67%	32	64
	2,874,055,680	94,371,840	3.28%	64	128
RoBERTa	126,491,481	1,794,048	1.4%	16	32
	128,285,529	3,588,096	2.80%	32	64
	131,873,625	7,176,192	5.44%	64	128
BERT Large	339,925,818	4,751,360	1.40%	16	32
	344,677,178	9,502,720	2.76%	32	64
	354,179,898	19,005,440	5.37%	64	128
BERT	111,308,346	1,794,048	1.61%	16	32
	113,102,394	3,588,096	3.17%	32	64
	116,690,490	7,176,192	6.15%	64	128

F.2 Grid search results

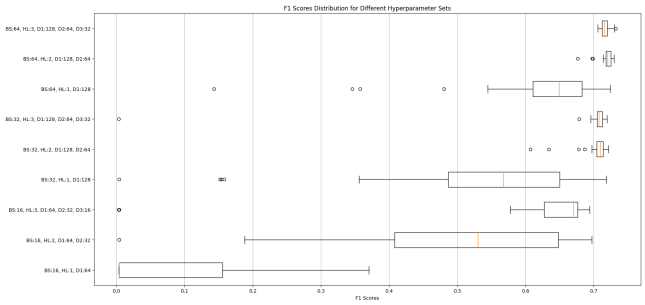
We conduct a grid search to optimize the hyperparameters of our experiment. These hyperparameters include the batch size for training, the number of hidden layers of the model, and the number of hidden dimensions. The resulting optimal configurations for the hyperparameters are displayed in Table 13. Figure 14 shows the complete box plot for these experiments for BERT, LLaMA and GPT.

Table 13: Best hyperparameters for each model

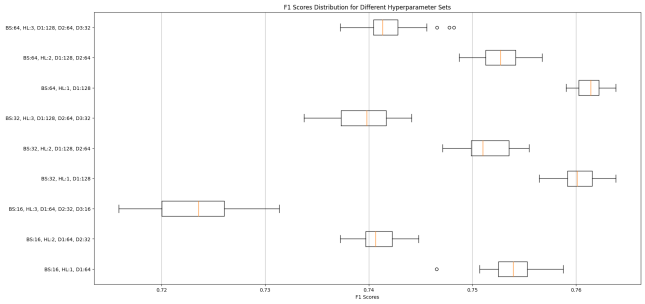
Model	Embedding dimensions	BEST CONFIGURATION		
		Batch size	Hidden layers	HD dims
BERT	768	32	1	128
BERT Large	1,024	64	1	128
RoBERTa	768	64	1	128
Phi	2,560	64	2	128-64
LLaMA	4,096	64	2	128-64
GPT	1,536	64	1	128



(a) BERT



(a) LlaMA



(b) GPT

Figure 14: Grid search.

F.3 Resource consumption

Additionally, we measured the resource consumption (power, energy and memory usage as well as training time) of this LoRA fine-tuning and display the results in Table 14.

While larger models typically require more resources, this is not a strict rule across all models. For example, BERT consumes more power, energy, and memory than BERT Large, indicating that model architecture and implementation efficiency also play significant roles in resource consumption.

Table 14: Model power consumption, energy usage, memory utilization and training time.

Model	Power Usage (W)	Energy usage (KWh)	Memory Utilization (GB)	Training Duration (Hours)
LLaMA	761.74	19.69	25.73	95.30
Phi	895.94	26.24	29.16	91.65
BERT Large	457.17	1.53	3.34	13.24
RoBERTa	534.45	0.90	1.68	49.32
BERT	507.07	0.91	1.78	41.85

G Weighted-Average Performance Metrics

To complement the macro-averaged results presented in Section 10.3, we additionally report the weighted-average performance metrics in Table 15. Including these results enables a complete comparison across models and offers additional insight into how each architecture behaves when class prevalence is taken into consideration.

Table 15: Weighted average performance when comparing the original test set and the new test set described in Sec 10.3

	Previous			After		
Model	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
TextCNN	75.69%	69.67%	70.58%	84.33%	68.38%	69.86%
BiGRU-TextCNN	76.49%	69.38%	70.38%	86.44%	72.74%	76.55%
Our approach	78.57%	74.90%	75.07%	87.79%	64.21%	69.22%

H Threshold Analysis

To assess the influence of the threshold on classification performance, we conducted a systematic evaluation of our two-step neural network approach across a wide range of threshold settings. We examined thresholds slightly below and above our primary choice, as well as substantially lower and higher thresholds. Additionally, we included a fully unfiltered setting in which all available CWEs were retained. The results are summarized in Table 16.

Table 16: Comparison improvement of 2-step neural network vs 1-step-neural network using LLaMA as a vectorizer on macro avg

Model	**num of classes**	**Precision**	**Recall**	**F1-score**
1-step	124 (all)	27.20%	37.10%	26.85%
	91	48.94%	53.27%	48.47%
	62	54.38%	62.72%	56.63%
	57	55.48%	61.08%	55.07%
	52	60.20%	62.79%	59.35%
	26	66.30%	74.21%	67.81%
2-step nn	124 (all)	31.79%	41.83%	32.12%
	91	55.31%	57.06%	54.64%
	62	58.22%	65.19%	59.84%
	57	59.64%	61.66%	58.82%
	52	60.34%	64.94%	60.75%
	26	66.51%	73.46%	68.27%

I Concept drift

Figure 15 illustrates the year-over-year CWE frequency comparison between 2022 and 2023. As the figure shows, there is no clear indication of concept drift during this period. This stability reinforces our decision not to base the threshold selection on temporal trends, as the absence of a meaningful inflection point would make such a criterion subjective rather than data-driven.

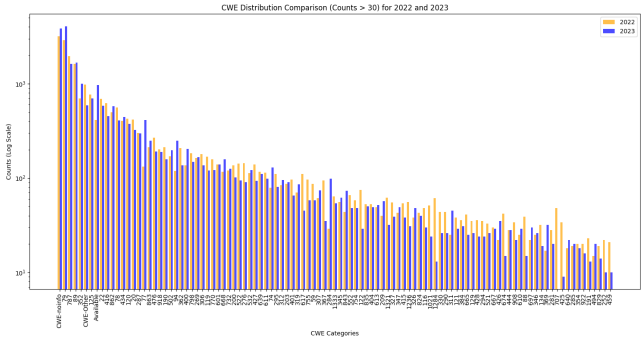


Figure 15: Concept drift in CWE across 2022 and 2023

J Coarse-grained improvement attempt

As mentioned in Section 9, the coarse-grained step of our two-step CWE classification pipeline has the greatest impact on overall performance. To improve this step, we experimented with fine-tuning the LLaMA model specifically for CWE parent prediction, in contrast to our standard approach of fine-tuning on CWE child (base/variant) predictions. The underlying hypothesis was that embeddings optimized for predicting CWE parents might be more specific and relevant for that level of classification. We compared two strategies:

- **Fine-tuned for class (parent)**: Llama model is fine-tuned in predicting the CWE parent from the CVE description.
- **Fine-tuned for base/variant (child)** Llama model is fine-tuned in predicting the CWE child from the CVE description (same as Section 6). To compare with parent prediction, we traced the corresponding CWE class during inference.

For example, consider CVE-2018-25018, which describes an "out-of-bounds write" and maps to CWE-787, a subtype of CWE-119. In the first configuration, the model is fine-tuned to directly map the CVE to the broader CWE-119. In the second configuration, the model is fine-tuned to predict the specific CWE-787, which is then traced back to its parent, CWE-119..

Table 17: Model performance of mapping CVE descriptions to CWE classes (parents) on the weighted average

Model	**Precision**	**Recall**	**F1-score**	**Accuracy**
Fine-tuned for class	84.81%	84.27%	83.81%	84.27%
Fine-tuned for base/variant	84.45%	84.14%	83.80%	84.14%

Table 17 summarizes the performance of both approaches using weighted averages. The results are comparable, indicating that fine-tuning for CWE parent prediction does not offer any significant improvement over fine-tuning for CWE child prediction. This finding demonstrates that focusing on fine-grained CWE base/variant predictions (with subsequent backtracking to the parent) is sufficient to capture the necessary distinctions in CVE descriptions for accurate CWE classification. Moreover, it suggests that the embeddings learned at the base/variant level carry enough semantic information to generalize effectively across both levels, supporting a streamlined two-step classification approach without the need for additional fine-tuning specifically for class-level distinctions.