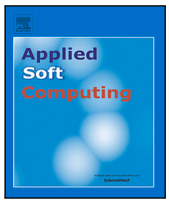
Applied Soft Computing Journal 182 (2025) 113612

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc



Enhancing long-tailed software vulnerability type classification via adaptive data augmentation and prompt tuning

Long Zhang [a], Xiaolin Ju [a,*], Lina Gong [b,*], Jiyu Wang [a], Zilong Ren [a]

[a] School of Artificial Intelligence and Computer Science, Nantong University, Nantong, China
[b] School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China

ARTICLE INFO

Keywords:
Software vulnerability type classification
Long-tailed learning
Prompt tuning
Feature fusion

ABSTRACT

Software vulnerability type classification (SVTC) is essential for efficient and targeted remediation of vulnerabilities. With the rapid increase in software vulnerabilities, the demand for automated SVTC approaches is becoming increasingly critical. However, the SVTC is significantly affected by the long-tailed issues, where the distribution of vulnerability types is highly unbalanced. Specifically, a small number of head classes contain a large volume of samples, while a substantial portion of tail classes consists of only a limited number of samples. This imbalance poses a significant challenge to the classification accuracy of existing approaches. To alleviate these challenges, we propose an innovative approach VulTC-LTPF, which integrates prompt tuning with long-tailed learning to enhance the effectiveness of SVTC. Within VulTC-LTPF, an adaptive error-rate-based data augmentation strategy is developed. This strategy allows the SVTC model to dynamically augment data for tail classes types with limited sample size during training, thereby mitigating the impact of the long-tailed problem. Furthermore, VulTC-LTPF employs a hybrid prompt tuning strategy, aligning the training process more closely with pre-training, which enhances adaptability to downstream tasks. Unlike existing approaches that rely solely on either vulnerability description or source code, VulTC-LTPF leverages both sources of information. By incorporating a combination of hard and soft prompts, it facilitates a more comprehensive and effective classification strategy. Experimental results demonstrate that VulTC-LTPF achieves substantial performance improvements over four state-of-the-art SVTC baselines, with gains ranging from 26.1% to 55.1% in MCC. Ablation studies further validate the effectiveness of the adaptive data augmentation, prompt tuning, the integration of two types of vulnerability information, and the use of hybrid prompts. These findings highlight that VulTC-LTPF represents a promising advancement in the field of SVTC, offering significant potential for further progress in addressing software vulnerability type classification challenges.

1. Introduction

As software systems grow increasingly complex, vulnerabilities inherent within these systems have emerged as critical threats to software security [1]. These vulnerabilities are often exploited by malicious actors, leading to potentially severe consequences such as data breaches, financial losses, and the compromise of system integrity and security [2]. Moreover, the continuous expansion in the scale and complexity of software systems has resulted in a corresponding increase in the diversity and prevalence of vulnerabilities [3], presenting unprecedented challenges for the maintenance of software security. Therefore, it is crucial to promptly and accurately identify and fix vulnerabilities to ensure system security and integrity.

In the vulnerability fixing process, software vulnerability type classification (SVTC) is a key step that helps developers effectively identify the specific type of vulnerability, which provides an important basis for subsequent fixing work. First, accurately classifying vulnerability types enables the prioritization of remediation efforts, allowing for a focus on vulnerabilities that pose the greatest threat to the system. Second, identifying specific vulnerability types facilitates the implementation of more targeted and effective remediation strategies. For example, among the 25 most dangerous vulnerabilities [4] identified by the Common Weakness Enumeration (CWE), the highest-ranked is CWE-787 [5], which pertains to out-of-bounds write. This vulnerability represents one of the most prevalent and severe issues associated with memory operations. If such risk can be identified early during the vulnerability remediation process, developers can prioritize implementing boundary checks for memory operations in their code, thereby significantly mitigating system risk.

* Corresponding authors.
E-mail addresses: longzhang1219@gmail.com (L. Zhang), ju.xl@ntu.edu.cn (X. Ju), gonglina@nuaa.edu.cn (L. Gong), jyu.wang@outlook.com (J. Wang), zilongren23@gmail.com (Z. Ren).

https://doi.org/10.1016/j.asoc.2025.113612
Received 7 February 2025; Received in revised form 12 June 2025; Accepted 8 July 2025
Available online 22 July 2025

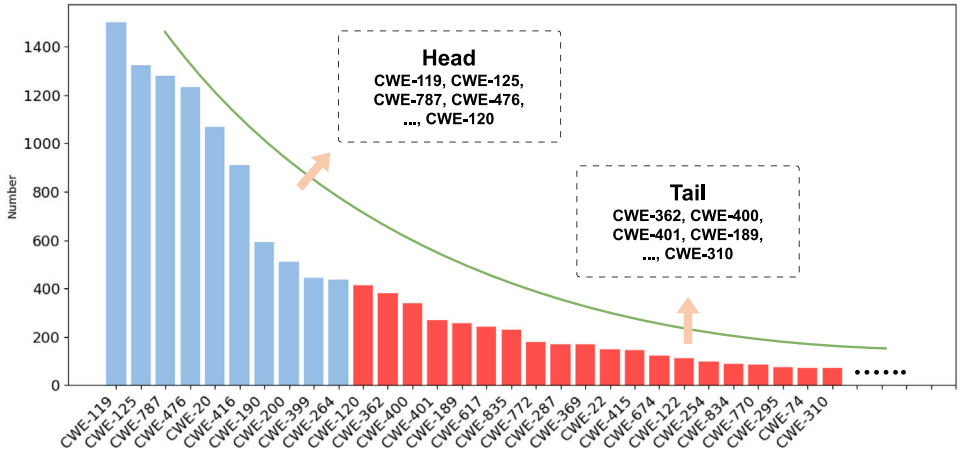


Fig. 1. Statistical analysis of vulnerability data from 2006 to 2024. The CWE types of vulnerabilities show a long-tailed distribution.

While existing research predominantly focuses on vulnerability detection [6–9], the critical task of SVTC remains understudied despite its direct impact on remediation efficacy. To address this gap, we conducted a statistical analysis of vulnerability data from 2006 to 2024 using open-source databases. The findings, illustrated in Fig. 1, reveal a distinct long-tailed distribution of vulnerability types. In this distribution, a small subset of types refer to as "head classes", accounts for the majority of samples, while the vast majority of vulnerability types, known as "tail classes", are represented by a limited number of samples. This long-tailed distribution can seriously affect the performance of the SVTC models. Specifically, it leads to insufficient learning of the tail classes during model training, adversely affecting classification accuracy and generalization ability. Tail classes, due to their scarcity, are often overlooked by the model, resulting in reduced classification accuracy. Moreover, in practical applications, tail class vulnerabilities frequently correspond to high-risk vulnerabilities. For instance, vulnerabilities such as CWE-22 and CWE-415 have a Common Vulnerability Scoring System (CVSS) score of 7.5 or higher, indicating a severity rating of "high" [10]. Consider CWE-22(Path Traversal) [11] as an example. This vulnerability enables attackers to manipulate file paths to access unauthorized files, potentially leading to the disclosure of sensitive information or the execution of malicious operations. Therefore, addressing the long-tailed distribution problem is critical for improving the performance of the SVTC model. Improving the model's ability to accurately classify tail class vulnerabilities not only boosts overall classification performance but also ensures that high-risk vulnerabilities are effectively identified and mitigated.

Previous SVTC models [12–15] rarely fully consider the long-tailed problem in the vulnerability type dataset. Since a few common head classes dominate the number of samples, this oversight causes the model to tend to learn the characteristics of the head classes during training, which seriously affects the accuracy of the model in identifying tail classes. While some studies [12,16] have proposed strategies to mitigate the long-tailed problem, they remain limited in scope. For instance, Wen et al. [17] introduced the LIVABLE approach, incorporating an adaptive re-weighting module. This module dynamically adjusts the loss weight during training based on the number of training rounds and the distribution of samples. However, this approach primarily focuses on adjusting the model's learning bias between head and tail classes across different training stages. It does not directly address the insufficient learning of tail class samples within each training iteration, leaving room for optimization. In contrast, our proposed approach introduces adaptive data augmentation that dynamically augments the tail class dataset by studying the model's classification efficiency of tail class samples during training and combining it with data augmentation techniques [18]. By dynamically increasing the number of tail classes samples during training based on model performance, the model can be encouraged to allocate more attention to underrepresented tail classes, thereby alleviating the long-tailed problem and improving overall classification performance. By dynamically increasing the number of samples of the tail classes according to the training performance during training, the model can be encouraged to allocate more attention to the tail classes in need during learning, thereby alleviating the long tail problem and improving the overall classification performance.

Currently, most SVTC models usually rely on a single type of vulnerability information for classification, such as the source code of the vulnerability [12,15,19,20] or the vulnerability description [13,14,21]. However, relying solely on one modality limits the model's ability to capture the full semantic features of vulnerabilities. In contrast, bimodal information combining source code and vulnerability description can significantly improve the accuracy of SVTC. Unlike some existing SVTC research approaches, using a pre-trained language model (PLM) in combination with fine-tuning paradigms [22,23] provides a new direction for improving SVTC performance. PLMs, trained on large-scale corpora, capture rich semantic information and complex language patterns, which can be effectively utilized for downstream tasks through fine-tuning [24–26]. However, SVTC tasks differ significantly from traditional pre-training tasks, which may hinder PLMs from effectively capturing domain-specific semantic information. First, the general semantic knowledge acquired during pre-training may be insufficient to meet the specific requirements of SVTC tasks directly. Second, fine-tuning PLMs typically necessitates a substantial amount of high-quality data, which is often scarce in SVTC tasks. These challenges underscore the need for a tailored approach to harness the potential of PLMs for SVTC tasks fully.

Building on the aforementioned motivation, we propose the VulTC-LTPF approach, which leverages prompt tuning and incorporates adaptive data augmentation. During the training phase, bi-modal information of the vulnerability is used as input, and a new input is constructed by integrating a hybrid prompt. The PLM CodeT5 [27] is then fine-tuned using the prompt tuning approach to achieve the SVTC task. At the end of each training round, the size of the required data augmentation is dynamically determined and the corresponding data augmentation is applied specifically to tail class categories. In the model prediction phase, the source code and vulnerability description of the target vulnerability are input into the fine-tuned model, which process the data using the trained hybrid prompt template. Finally, a verbalizer maps the model's predicted tokens to their corresponding vulnerability types. To evaluate the effectiveness of our proposed VulTC-LTPF approach, we compared it against four state-of-the-art SVTC baselines [15,28,29] (i.e., $\text{VulExplainer}_{\text{CodeBERT}}$, $\text{VulExplainer}_{\text{CodeGPT}}$, Devign, and ReGVD). We conducted a comprehensive evaluation of the model's

performance using standard evaluation metrics, including accuracy, precision, recall, F1 score, and Matthews Correlation Coefficient (MCC). The experimental results demonstrate that the VulTC-LTPF approach consistently outperforms the baseline models across all metrics. Notably, on the MCC metric, the VulTC-LTPF approach achieves a significant performance improvement, ranging from 26.1% to 55.1%. Furthermore, ablation studies confirm the critical contributions of bi-modal information, hybrid prompt templates, and adaptive data augmentation in enhancing the performance of the VulTC-LTPF approach.

The findings of this study highlight three promising directions for future research in SVTC that warrant further investigation. First, developing more effective strategies to address the long-tailed problem in SVTC datasets remains a critical area of focus. Second, exploring advanced prompt tuning-based methods offers significant potential for enhancing the performance of SVTC tasks. Third, delving deeper into additional sources of information related to software vulnerabilities, such as the structural features of source code, presents an opportunity to further improve the effectiveness of SVTC approaches.

The novelty and contributions of our study can be summarized as follows:

- **Dataset.** We refined and update the dataset to enhance its adaptability to the SVTC task. The specific processing steps will be discussed in detail in Sections 3.1 and 4.2.
- **Perspective.** We applied prompt tuning and long-tailed learning techniques to the task of software vulnerability type classification, achieving a significant and reliable improvement in performance.
- **Approach.** We propose the VulTC-LTPF approach, which combines adaptive data augmentation with bi-modal inputs (source code and vulnerability descriptions) into prompt tuning framework to optimize the SVTC task.
- **Practical Evaluation.** We conducted a comprehensive evaluation of VulTC-LTPF, assessing the effectiveness of our approach by comparing it against state-of-the-art SVTC baseline methods. Furthermore, we performed extensive multi-group ablation experiments to validate the robustness and efficacy of the proposed method.

Open Science. To promote open science and reproducible research, we share datasets, code, and detailed experimental results.[1]

Paper Organization. Section 2 is the background of this paper, introducing software vulnerability type classification, long-tailed learning, and prompt tuning. Section 3 provides a detailed explanation of the VulTC-LTPF framework, systematically outlining each stage of the proposed approach. Section 4 describes the experimental setup, including the research questions and their design motivations, the experimental object, the baseline approaches as well as the evaluation metrics. Section 5 presents the experimental results and their analysis, covering comparisons with baselines and ablations experiments. Section 6 discusses the impact of pre-trained language models and scaling factors on performance, while summarizing validity threats. Section 7 reviews related work and highlights the paper's innovations. Finally, Section 8 summarizes our findings and discusses future directions.

2. Background

This section begins by providing an overview of software vulnerability type classification, followed by an introduction to long-tailed learning and prompt tuning.

[1] https://github.com/ntu-juking/VulTC-LTPF.

2.1. Software vulnerability type classification

Common Weakness Enumeration (CWE) is a publicly available and widely used official vulnerability database that plays a vital role in addressing software vulnerabilities. Maintained by MITRE [30], the CWE provides a systematic classification, detailed descriptions, and an analysis of the potential impacts of various software weaknesses on security and quality. Through standardized vulnerability descriptions and classifications, CWE facilitates the accurate identification and reporting of vulnerabilities by security professionals, offers evidence-based recommendations for remediation, and assists in determining the type and severity of vulnerabilities. In the process of vulnerability remediation, this information aids in the establishment of appropriate priorities, thereby enhancing the efficiency of the remediation process.

However, SVTC is a challenging task [14,17] that requires security experts to manually analyze code to identify vulnerability types. But, there are some obvious problems with this process. First, manual analysis is highly dependent on the security expert's specialized experience and domain knowledge. Second, the manual classification process is time-intensive and resource-consuming. These issues make manual analysis inadequate for meeting the demands of large-scale vulnerability classification. For instance, the US National Vulnerability Database (NVD) recorded 28,902 vulnerabilities in 2023, with 4113 cases remaining unclassified [12]. Therefore, there is an urgent need for an efficient automated software vulnerability type classification tool that can quickly and accurately predict the potential vulnerability type, providing valuable reference for security experts to follow up on software vulnerability repair work.

2.2. Long-tailed learning

The long-tailed learning approach aims to solve the problem of severe class imbalance in data distribution. In many real-world tasks, the data set usually has a long-tailed distribution [31], that is, most data samples are concentrated in a few "head class" categories, while the vast majority of categories have a small number of samples and are "tail classes". The long-tailed problem is particularly prominent in the field of SVTC.

Long-tailed learning has become an important research direction in the field of computer vision [31–33]. There are some popular approaches to alleviate the long-tailed problem, such as data resampling [34], loss function re-weighting [35], and data augmentation [36]. Data resampling balances a dataset by adjusting the number of samples in different categories in the training data, usually by oversampling the tail classes or undersampling the head classes. Loss function re-weighting makes the model pay more attention to these difficult-to-classify samples by assigning higher weights to samples in the tail classes in the loss function. Data augmentation generates new samples by transforming the original data, thereby increasing data diversity and improving the generalization ability of the model.

These approaches have been widely used in the field of computer vision and have effectively alleviated the long-tailed problem. However, research on long-tailed distributions for the SVTC task is still limited. Therefore, based on the research and analysis of existing long-tailed learning approaches, we propose an effective long-tailed learning approach suitable for SVTC.

2.3. Prompt tuning

Prompt tuning is a technique in Natural Language Processing (NLP) aimed at optimizing model performance in downstream tasks by refining the prompts that guide pre-trained language models (PLMs) [37]. Unlike traditional fine-tuning, which updates model weights, prompt tuning enhances task adaptation by designing effective prompt templates. In the SVTC task, this approach helps PLMs better capture vulnerability-related information and distinguish between source code

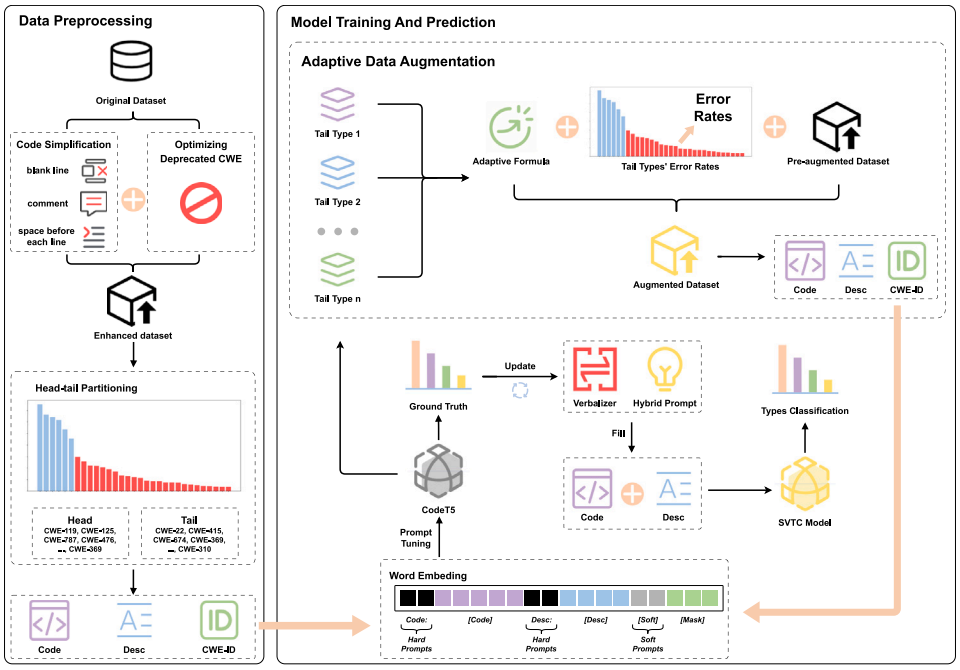


Fig. 2. Framework of VulTC-LTPF.

and vulnerability descriptions. Specifically, the template $f_{\text{prompt}}(x)$ transforms the original input x into a new input x' [38]. The template contains two main slots, one for filling in the input data and the other for filling in the expected answer, corresponding to data entry and answer prediction, respectively. The verbalizer maps the words predicted by the model to specific vulnerability type labels. Through these one-to-one or one-to-many mappings, semantically similar words are grouped into the same category.

Prompt templates can be classified into hard prompts and soft prompts [20]. Hard prompts are manually designed static templates that guide PLMs through specific tasks. In contrast, soft prompts consist of learnable embeddings added to input tokens, which can be optimized during training via backpropagation. Soft prompts are represented as continuous vectors, enabling the model to adapt and fine-tune them for specific tasks.

In the SVTC task, natural language is added to the input to construct prompt templates that combine source code and vulnerability descriptions.

A study by Wang et al. [39] applied prompt tuning to a variety of code intelligence tasks, including defect prediction, code summarization, and code translation. Their experimental results show that prompt tuning performs well in these tasks. Similarly, Yang et al. [40] applied prompt tuning to the Stack Overflow title generation task with satisfactory results. These results show that prompt tuning has significant performance advantages over traditional fine-tuning approaches in multiple domains. In this study, we apply prompt tuning to the SVTC task and explore its effectiveness in depth to verify the potential and advantages of this approach in vulnerability types classification.

3. Approach

Fig. 2 shows the framework of our proposed VulTC-LTPF approach. It is divided into three main phases, namely the data preprocessing phase, the model training phase and the model prediction phase. In the remainder of this section, we describe these three phases in detail.

3.1. Data preprocessing phase

In our study, we chose the dataset MegaVul shared by Ni et al. [41]. as the initial dataset and selected vulnerability functions in the C/C++ programming language. Through in-depth analysis of the MegaVul dataset, we found that certain elements in the exploit code (e.g., blank lines, comments, and spaces before paragraphs) may adversely affect the performance of the model. Specifically, these elements can take up the length of the input of a pre-trained language model due to its length limitation, and in the case of comments, the model may incorrectly recognize them as code. To optimize model performance, we remove these redundant elements to ensure that the model can focus on the actual functional part of the code and learn as much relevant information as possible. The simplified version of the code does not affect its functionality. These preprocessing steps are intended to help make the source code more suitable as input for SVTC tasks.

Subsequently, according to our research findings in Section 4.2, some vulnerability entries in the original dataset used CWE-IDs that had been deprecated on the official website. For this reason, we optimized these entries by deleting or updating the vulnerability entries. After these processes, we obtained an enhanced dataset that is more suitable for the SVTC task. We then divided the dataset into head and tail classes.

3.2. Model training phase

In the model training stage, we first consider using the bi-modal information of vulnerabilities (i.e., source code and vulnerability description) to construct prompt templates. We apply prompt tuning to the pre-trained model CodeT5 to perform the downstream task of SVTC. Then, considering the long-tailed distribution of vulnerability types in the dataset and the data imbalance problem, we designed an adaptive data augmentation module that adaptively augments the data for each tail class based on the model's predicted performance on these tail classes. In the framework of our approach, the two components of adaptive data augmentation and prompt tuning are closely connected and

complement each other. The effectiveness of prompt tuning depends mainly on the quality and balance of the input data, while adaptive data augmentation provides a fairer training basis for prompt tuning. In turn, the prompt tuning fully uses the enhanced sample distribution to improve further the model's performance regarding semantic modeling and category discrimination. The two work together to improve the recognition ability of tail classes and the overall classification accuracy.The model training part is described in detail below.

3.2.1. Prompt template and verbalizer construction

Prompt tuning approaches utilize prompt templates to modify raw inputs and generate a new input. The design of prompt templates is a critical and challenging task that often requires multiple trials and optimizations to produce high-quality templates for specific downstream tasks. Prompt templates can be classified into three types: hard prompts, soft prompts, and hybrid prompts.

Hard Prompt. Hard prompts [22,42] refer to manually designed natural language phrases or templates that are usually fixed before model training and remain unchanged throughout the training process. Hard prompt templates have two types of slots: input slots and answer slots. For the SVTC task, considering that our approach uses bi-modal information as input, two input slots are required. The specific hard prompt template structure can be expressed as follows:

$$f_{hard} = \text{The code snippet: [X] The vulnerability description: [Y] Classify the vulnerability type: [Z]} \quad (1)$$

Here, input slot [X] will be filled with source code and input slot [Y] with a vulnerability description. PLM predicts the probability distribution of the label term at position [Z] based on the given input information, and the label term with the highest probability is used as the intermediate answer generated by PLM.

Soft Prompt. Soft prompts [22,38,42,43] are a learnable form of embedding representation that allows optimization during training. The greatest advantage of soft prompts is their flexibility, as they can be automatically adjusted according to the specific requirements of the task, thereby enabling the model to better understand the input and perform specific downstream tasks. For SVTC, the specific soft prompt template can be defined as:

$$f_{soft} = \text{[SOFT] [X] [SOFT] [Y] [SOFT] [Z]} \quad (2)$$

In this template, we initialize the [SOFT] tokens using the natural language embeddings from the hard prompt above. Specifically, the first [SOFT] token is initialized to "The code snippet:", the second [SOFT] token is initialized to "The vulnerability description:", and the third [SOFT] token is initialized to "Classify the vulnerability type:".

Hybrid Prompt. Hybrid prompting [40,42] combines the advantages of both hard and soft prompting. Specifically, hard prompts are suitable for denoting important information that is closely related to downstream tasks because they remain unchanged during training. For example, in the SVTC task, "The code snippet:" and "The vulnerability description:" are important markers, so we retain these as hard prompt tokens. In contrast, soft prompt tokens can be optimized throughout the training process and are more flexible, so we use them to retain the token "Classify the vulnerability type:", because there are many alternative tokens for this phrase, such as "Classify the CWE:" or "The vulnerability type is:". The use of soft prompts allows the model to dynamically adjust its wording based on the context, thereby augmenting the diversity and adaptability of the prompt. For the SVTC task, the hybrid prompt can be designed as follows:

$$f_{hybrid} = \text{The code snippet: [X] The vulnerability description: [Y] [SOFT] [Z]} \quad (3)$$

In our research, we found that using a hybrid prompt (i.e., Eq. (3)) enables VulTC-LTPF to achieve optimal performance. For a detailed analysis, see our ablation study section.

Verbalizer. Verbalizer is a key component in the prompt tuning approach. It is responsible for mapping the predictions generated by the model at the [MASK] position to specific vulnerability type labels. In this process, the output label words of the model are restricted by a vocabulary that maps the generated words to the labels of the vulnerability types in the actual task. One label for a specific category can have one or more label words. Depending on the requirements, Verbalizer can be divided into the following two types: one-to-one Verbalizer and one-to-many Verbalizer. In the SVTC task, we designed one-to-one and one-to-many Verbalizers suitable for the task, as follows:

$$\text{Verbalizer}_{O2O} = \begin{cases} \text{“}CWE-119\text{”}: & [\text{“}buffer overflow\text{”}] \\ \text{“}CWE-125\text{”}: & [\text{“}out-of-bounds read\text{”}] \\ \cdots\cdots & \\ \text{“}CWE-310\text{”}: & [\text{“}cryptographic issue\text{”}] \end{cases} \quad (4)$$

$$\text{Verbalizer}_{O2M} = \begin{cases} \text{“}CWE-119\text{”}: & [\cdots + \text{“}memory violation\text{”}] \\ \text{“}CWE-125\text{”}: & [\cdots + \text{“}information leak\text{”}] \\ \cdots\cdots & \\ \text{“}CWE-310\text{”}: & [\cdots + \text{“}insecure cryptography\text{”}] \end{cases} \quad (5)$$

In our study, we found that the verbalizer designed in Eq. (5) enables the best performance of VulTC-LTPF. For a detailed analysis, please see our ablation study section.

3.2.2. Prompt tuning on CodeT5

The Cross-Entropy(CE) Loss [44] Function is widely used in multi-class classification tasks, which effectively measures the discrepancy between the model predictions and the true labels, and thus drives the optimization of the model parameters. Therefore, in the model training phase, in order to optimize the performance of the model in the SVTC task, we choose to use the Cross-Entropy loss function. For each training sample, the loss function is calculated as follows:

$$\mathcal{L}_{\text{CE}} = -\sum_{i=1}^{n} y_i \log(p_i) \quad (6)$$

where n is the number of categories, y_i is the indicator function of the true label, and p_i is the output predicted by the model.

In VulTC-LTPF we have chosen the pre-trained CodeT5 model for prompt tuning. This is mainly because CodeT5 is specially designed to handle source code related tasks and performs well in a variety of code understanding tasks. In addition, studies by Ruan et al. [45] and Niu et al. [46] have verified the powerful performance of CodeT5 in different downstream tasks, demonstrating its versatility and reliability.

3.2.3. Adaptive data augmentation

According to our study of the SVTC dataset, we found that the types of vulnerabilities in the dataset show a long-tailed distribution, which can lead to data imbalance problems. To address the long-tailed problem in the SVTC task, we designed adaptive data augmentation in VulTC-LTPF, as detailed in Algorithm 1.

The adaptive data augmentation module is designed to improve the model's classification performance for tail classes. This module dynamically evaluates the error rate of tail classes on the validation set after each training round, and determines the number of augmented samples to be generated based on these error rates. Then, tail classes are augmented adaptively using multiple data augmentation approaches. Finally, the augmented training set is generated for subsequent training. This approach efficiently augments the tail class sample by dynamically adjusting the augmentation strength of the tail class sample, which can effectively improve the negative impact caused by data imbalance and improve the model's classification ability for the tail class. The details of this module are described in detail below.

Algorithm 1 Adaptive Data Augmentation

Input: Training set D_{train}, Validation set D_{val}, Trained model M, Scaling factor K, Set of augmentation approaches S
Output: Augmented dataset D_{aug}

1: Initialize $D_{aug} \leftarrow D_{train}$
2: Initialize $ErrorRates \leftarrow \emptyset$
3: Initialize $AugmentationCounts \leftarrow \emptyset$
4: **for** each class $i \in C_{tail}$ **do**
5: $N_i \leftarrow$ number of samples of category i in D_{val} where $Y_i = i$
6: $T_i \leftarrow$ number of samples correctly predicted as category i by model M
7: **if** $N_i > 0$ **then**
8: $ErrorRate_i \leftarrow 1 - \frac{T_i}{N_i}$
9: **else**
10: $ErrorRate_i \leftarrow 0$
11: **end if**
12: $ErrorRates[i] \leftarrow ErrorRate_i$
13: **end for**
14: **for** each class $i \in C_{tail}$ **do**
15: $A_i \leftarrow \lfloor K \times ErrorRates[i] \rfloor$
16: $AugmentationCounts[i] \leftarrow A_i$
17: **end for**
18: **for** each class $i \in C_{tail}$ **do**
19: $Samples_i \leftarrow$ select A_i samples from D_{train} with category i
20: **for** each sample X_i in $Samples_i$ **do**
21: $AugmentedSample \leftarrow$ randomly apply one approach from S to X_i
22: $D_{aug} \leftarrow D_{aug} \cup \{(AugmentedSample, Y_i)\}$
23: **end for**
24: **end for**
25: **return** D_{aug}

Evaluate the error rate of the tail class and determine the augmentation scale. In order to achieve adaptive data augmentation, the model's performance on the tail class categories needs to be evaluated at the end of each training round, so we first need to quantify the error rate of the tail class categories (Lines 4–13). The error rate for the tail class i is calculated as follows:

$$\text{ErrorRate}_i = 1 - \frac{T_i}{N_i} \tag{7}$$

where N_i denotes the total number of samples in the validation set belonging to class i, while T_i denotes the number of samples that the model correctly predicts to belong to class i. These quantities are formally defined as follows:

$$T_i = \sum_{j=1}^{N_i} \mathbb{I}(\hat{y}_j = y_j \wedge y_j = i) \tag{8}$$

After obtaining the error rate of the model in each tail class category, we designed an adaptive data augmentation calculation formula to determine the scale of data augmentation required for each tail class category (Lines 14–17). The formula is as follows:

$$A_i = \lfloor k \times \text{ErrorRate}_i \rfloor \tag{9}$$

where k is a predefined scaling factor that determines the amplification rate of augmented samples.

To determine an appropriate value for the scaling factor k in adaptive data augmentation, we experimented with the grid search approach [47] (detailed in Section 6.1) using values ranging from 1 to 20. The results demonstrated that $k = 15$ provided the best balance between enhancing tail classes representation and avoiding overfitting. This empirical evaluation ensures the reproducibility and scalability of the proposed strategy.

Data Augmentation Approaches. To generate augmented samples, we choose to augment the source code of the vulnerability. We construct a data augmentation approach library that contains multiple data augmentation techniques. When the tail class category's augmentation scale is determined, each category will perform a determined number of data augmentations. Each time, a data augmentation approach is randomly selected from the approach library and applied to the original sample (Lines 18–24). In this way, we can effectively increase the diversity of augmented samples and improve the robustness of the model. In the data augmentation approach library, we have designed the following data augmentation approaches:

- **Variable name substitution.** This approach identifies variable names in code snippets through regular expressions and replaces them with randomly generated new names. This approach increases the divsamples' diversity without changing the code's semantics augmenting the model's ability to handle potentially diverse inputs.
- **Insertion of redundant code.** This approach simulates the additional non-functional parts that may exist in the actual code by inserting non-functional code at random locations. In this way, the complexity and diversity of the training samples are increased.

3.3. Model prediction phase

In the model prediction stage, we combine the bi-modal information of the vulnerability with the hybrid prompt template we designed as an input, and then input it into the trained SVTC model for software vulnerability type classification.

4. Experimental setup

In this section, we first describe the research questions and their design motivations. We then detail the experimental subject, baseline, evaluation metrics, and experimental settings in turn.

4.1. Research questions

To demonstrate the competitiveness of VulTC-LTPF and justify its component configurations, we formulate the following five research questions (RQs) for our study.

RQ1: What is the performance of VulTC-LTPF in SVTC?

Motivation: In RQ1, the goal of our study is to evaluate the overall effectiveness of the VulTC-LTPF approach in solving the SVTC task. To this end, five automated performance evaluation metrics (i.e., Accuracy, Precision, Recall, F1 score, and MCC) were used to comprehensively measure its classification performance.

RQ2: What is the impact of adaptive data augmentation on the performance of the VulTC-LTPF?

Motivation: Adaptive data augmentation aims to alleviate the problem of long-tailed distribution of the dataset and optimize the training process of the model by enhancing the data quality and diversity of the samples in the tail class. A study of its impact on VulTC-LTPF performance will not only help to demonstrate the effectiveness of the strategy, but also validate its actual contribution in enhancing the model capability.

RQ3: What is the impact of bi-modal information on the performance of the VulTC-LTPF?

Motivation: In our VulTC-LTPF approach, the two modal data, source code and vulnerability description, are fused through a prompt template we designed to form a new bi-modal information input into the model. Therefore, in RQ3, our research goal is to explore whether this bi-modal input configuration can achieve the best performance

Table 1
Comparison between Big-Vul and MegaVul.

Statistic	Big-Vul	MegaVul
Number of repositories	310	1062
Number of Vul function	10,547	17,380
Number of commits	4058	9288
Number of CVE IDs	3539	8476
Date range of crawled CVEs	2013/01~2019/03	2006/01~2024/04
Number of CWE IDs	92	176
Function extract approach/(Quality)	Lizard/(Low)	Tree-sitter/(High)
Code integrality	Partial	Full

of VulTC-LTPF. In addition, we would like to determine which input modality contributes the most to the performance of VulTC-LTPF by comparing different modal configurations (using only source code or only vulnerability description).

RQ4: What is the impact of the prompt tuning paradigm on the performance of the VulTC-LTPF?

Motivation: Our approach VulTC-LTPF conducts experiments based on prompt tuning, which capitalizes on the latent knowledge in the pre-trained model. In RQ4, the goal of our research is to explore the applicability of prompt tuning in SVTC tasks and the magnitude of its performance improvement over traditional fine-tuning approaches.

RQ5: What is the impact of different prompt settings on VulTC-LTPF performance?

Motivation: In prompt tuning approaches, the design of the prompts template and the verbalizer can directly affect the performance of the model. Settings such as the use of different prompt templates, the choice between soft and hard prompts, and the selection of the type of the verbalizer can significantly impact the performance of the model. Therefore, selecting the appropriate prompt template and verbalizer for downstream tasks is a challenging and open problem. In RQ5, we aim to explore the effects of different prompt settings and demonstrate that the hybrid prompt strategy adopted by VulTC-LTPF is the optimal solution. Additionally, we also investigate the impact of the number of mapping words in the verbalizer on the performance of the VulTC-LTPF approach.

4.2. Experimental subjects

This study uses the latest version of MegaVul [41] as its initial dataset. MegaVul contains 17,975 vulnerabilities from 1062 open source repositories, covering 176 different types of vulnerabilities disclosed between January 2006 and April 2024. In previous SVTC studies, the most widely used dataset was BigVul [48]. Compared to the BigVul dataset, MegaVul contains 64.9% more vulnerabilities and covers a wider range of vulnerability types. In addition, the BigVul dataset only contains vulnerability data from 2003 to 2019 and is of lower quality, with a number of problems including incomplete functions, incorrect function merging and missing commit information. Unlike BigVul, which uses regular expressions to extract functions, MegaVul uses a complex syntax rule-based parse tree to extract functions. The specific differences between these two datasets are shown in Table 1, and the tabular results indicate that MegaVul is more suitable than BigVul for SVTC tasks.

The MegaVul dataset provides a wealth of vulnerability information, including vulnerable code, descriptions, and vulnerability types. However, some vulnerability entries in MegaVul use deprecated CWE-IDs. For example, vulnerability CVE-2016-1640 has the vulnerability type CWE-17. However, according to the information on the official CWE website [49], it can be confirmed that this vulnerability type has been deprecated. In order to ensure the quality of the dataset and to make it better suited to the SVTC task, we have updated or removed data with this condition.

We ended up with an enhanced dataset that contains 13, 124 vulnerabilities. The contents of the dataset are shown in Table 2. We grouped the categories with a sample size of less than 70 into a new class called "Remain". The table shows the distribution of vulnerability types in the dataset, and it can be seen that the head and tail classes account for 74.1% and 25.9% of the samples, respectively. During the dataset partitioning process, we ensured that vulnerabilities with the same CVE ID (they have different source code snippets with the same descriptive information) are kept in the same dataset to maintain validity and fairness. We partitioned the data into non-overlapping training, validation and testing sets at a ratio of 8:1:1.

Table 2
The types of vulnerabilities in the experimental dataset and their corresponding proportions, as well as the distribution of the head and tail classes. Categories with a sample size of less than 70 are classified into the "Remain" class.

Types	Ratio	Group	Types	Ratio	Group
CWE-119	11.43%		CWE-835	1.76%	
CWE-125	10.09%		CWE-772	1.36%	
CWE-787	9.75%		CWE-287	1.29%	
CWE-476	9.40%		CWE-369	1.28%	
CWE-20	8.15%		CWE-22	1.13%	
CWE-416	6.95%	Head	CWE-415	1.12%	
CWE-190	4.51%		CWE-674	0.92%	
CWE-200	3.90%		CWE-122	0.85%	Tail
CWE-399	3.38%		CWE-254	0.74%	
CWE-264	3.35%		CWE-834	0.68%	
CWE-120	3.16%		CWE-770	0.66%	
CWE-362	2.89%		CWE-295	0.58%	
CWE-400	2.58%		CWE-74	0.55%	
CWE-401	2.06%	Tail	CWE-310	0.54%	
CWE-189	1.95%		Remain	1.14%	
CWE-617	1.86%				

4.3. Baselines

We compare our approach to the state-of-the-art SVTC model VulExplainer [15]. We also compare our approach to models designed for binary classification vulnerability detection tasks, including Devign [28] and ReGVD [29]. The baseline approach is described below:

- **VulExplainer**: The transformer-based hierarchical distillation model proposed by Fu et al. [15] for handling highly unbalanced CWE labels to improve the performance of SVTC. In this approach, the data distribution is made more balanced by grouping similar CWE-IDs based on CWE abstract types.
- **Devign**: The GNN-based approach for vulnerability detection proposed by Zhou et al. [28] The model first converts source code functions into graph structures, and then updates the node representations using Gated GNNs [50] to capture semantic and structural features in the source code. Finally, Devign uses a 1-D CNN-based pooling operation for vulnerability prediction. Although the authors of Devign [28] have not released an official implementation, we used the re-implementation provided by [29] and followed the training protocol of the original Devign for experimental validation.
- **ReGVD**: The GNN-based approach for source code vulnerability detection proposed by Nguyen et al. [29]. The approach represents a source code function as a flat sequence of tokens and constructs a graph based on these tokens, where the nodes of the graph are initialized by embedding vectors generated by a pre-trained programming language model. ReGVD learns graphical embeddings of the source code by introducing residual connectivity between the GCN [51] layers and combining it with pooling operations, which ultimately performs vulnerability prediction.

4.4. Evaluation metrics

In our study, we use five evaluation metrics: accuracy, precision, recall, F1 score, and Matthews correlation coefficient (MCC) to provide a comprehensive performance evaluation.

According to previous research, the first four metrics are more common for SVTC tasks, while MCC is particularly suitable for datasets with class imbalance problems. Since multiple vulnerability types need to be predicted, we use macro-averaged metrics to compute the final results. These metrics measure the performance of the model in multi-class classification from different perspectives, especially how well it handles class imbalance problems. Next, we describe in detail the approach to calculating these evaluation metrics.

TP: True Positive, which indicates the number of samples that were correctly categorized into the positive category. In the SVTC task, it indicates that the model correctly identifies samples with different types.

TN: True Negative, which indicates the number of samples that were correctly categorized as negative categories. For each type, it indicates the number of samples that were accurately categorized as not belonging to that type.

FN: False Negative, which indicates the number of samples in the positive category that were misclassified as being in the negative category. For each type, it indicates the number of samples that were misclassified as not belonging to that type.

FP: False Positive, which indicates the number of samples in the negative category that were misclassified as being in the positive category. For each type, it indicates the number of samples that were misclassified as belonging to that type.

For each type i, these metrics can be represented as: TP_i, TN_i, FN_i, FP_i.

Accuracy: Accuracy is the ratio of the number of correctly predicted vulnerability type samples to the total number of all predicted samples.

$$\text{Accuracy} = \frac{\sum_i TP_i}{\sum_i (TP_i + FP_i + FN_i + TN_i)} \tag{10}$$

Precision: Precision rate indicates the proportion of all samples predicted to be in the positive category that are actually in the positive category. Macro-precision rate is averaged over each type and is used to measure the overall precision of the model over all categories. Where N denotes the number of vulnerability types.

$$\text{Precision}_{macro} = \frac{1}{N} \sum_i \frac{TP_i}{TP_i + FP_i} \tag{11}$$

Recall: Recall indicates the proportion of samples that are correctly predicted to be positive out of those that actually fall into the positive category. Macro-recall is averaged over each type and is used to measure the overall recall ability of the model over all categories.

$$\text{Recall}_{macro} = \frac{1}{N} \sum_i \frac{TP_i}{TP_i + FN_i} \tag{12}$$

F1 score: The F1 score is a reconciled average of precision and recall and is used to assess the precision and recall of the model in aggregate. Macro-F1 score is the average of the F1 scores for each type.

$$\text{F1-score}_{macro} = \frac{1}{N} \sum_i 2 \times \frac{\text{Precision}_i \times \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{13}$$

MCC: The Matthews Correlation Coefficient is a metric that combines True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) and is particularly well suited for dealing with type imbalances. Macro MCC is the average of each type of MCC.

$$\text{MCC}_{macro} = \frac{1}{N} \sum_i \frac{TP_i \cdot TN_i - FP_i \cdot FN_i}{\sqrt{(TP_i + FP_i)(TP_i + FN_i)(TN_i + FP_i)(TN_i + FN_i)}} \tag{14}$$

For performance evaluation, we use macro versions of the metrics precision, recall, F1 score and MCC as default metrics for evaluating the performance of different approaches.

Table 3
Performance comparison between VulTC-LTPF and SVTC baselines, with the best results for each metric highlighted in bold.

Approach	Accuracy	Precision	Recall	F1	MCC
$\text{VulExplainer}_{\text{CodeBERT}}$	0.446	0.458	0.349	0.357	0.405
$\text{VulExplainer}_{\text{CodeGPT}}$	0.432	0.365	0.282	0.273	0.385
Devign	0.221	0.190	0.130	0.122	0.152
ReGVD	0.175	0.194	0.153	0.140	0.115
VulTC-LTPF	**0.688**	**0.640**	**0.642**	**0.620**	**0.666**

4.5. Experimental settings

In our experiments, we performed the following experimental settings:

Prompt template construction. To achieve prompt tuning, we use the OpenPrompt framework [52] and build hard templates, soft templates and hybrid templates through the *ManualTemplate*, *SoftTemplate* and *MixedTemplate* APIs.

Model hyperparameter configuration. In our experiments, we followed the default parameter configuration of CodeT5. Specifically, the word embedding dimension and hidden layer size are both set to 768, and the model contains 12 attention heads and 12 layers of Transformer encoders. The optimization process uses the AdamW optimizer [53], with an initial learning rate of 5e−5. During training, the batch size is set to 32, and the maximum length of the input sequence is 512. To avoid overfitting the model, an early stop strategy [54] is used, i.e., when the performance of the validation set does not improve for 10 epochs in a row, training is stopped, and the model with the best validation set performance is selected as the final model.

Input length analysis and truncation strategies. Due to the input length limitation of CodeT5, in order to maximize the retention of vulnerability information, we implemented truncation of the bi-modal input based on code simplification of the source code information in the dataset, where the maximum length of the code snippet was set to 384 words and the maximum length of the vulnerability description was set to 64 words.

Experimental environment. All experiments were performed on a computer equipped with an Intel(R) Core(TM) i5-13600K processor, a GeForce RTX 4090 GPU with 24 GB of graphics memory, and Windows 10 operating system.

5. Experimental results

5.1. RQ1: What is the performance of VulTC-LTPF in SVTC?

Approach: To evaluate the effectiveness of our approach, we adopt commonly used evaluation metrics (including accuracy, precision, recall, F1 score and MCC) and comprehensively compare them with four state-of-the-art baselines. In the experiment, the experimental settings of all baselines are kept consistent, and their respective optimized hyperparameters are used.

Results: As shown in Table 3, VulTC-LTPF significantly outperforms existing baseline approaches in all evaluation metrics, demonstrating its strong performance in the SVTC task. Specifically, VulTC-LTPF's accuracy metric improved by 24.2% to 51.3%, precision metric improved by 18.2% to 45%, recall metric improved by 29.3% to 51.2%, F1 score improved by 26.3% to 49.8%, and MCC metric improved by 26.1% to 55.1%.

The superior performance of VulTC-LTPF can be attributed to several key innovations. First, VulTC-LTPF employs adaptive data augmentation, which effectively mitigates the long-tailed problem and improves the classification on tail class vulnerability types. In contrast, other baseline approaches, do not employ a similar long-tailed learning approach. Second, VulTC-LTPF introduces prompt tuning, which allows the model to leverage the latent knowledge of the pre-trained model

Table 4
Comparative results between our approach VulTC-LTPF and VulTC-LTPF without ADA, with the best results highlighted in bold.

Approach	Accuracy	Precision	Recall	F1	MCC
w/o ADA	0.654	0.600	0.615	0.582	0.629
VulTC-LTPF	**0.688**	**0.640**	**0.642**	**0.620**	**0.666**

Table 5
Comparative results between our approach VulTC-LTPF and VulTC-LTPF using traditional data augmentation, with the best results highlighted in bold.

Approach	Accuracy	Precision	Recall	F1	MCC
w/o Adaptive	0.661	0.589	0.617	0.584	0.637
VulTC-LTPF	**0.688**	**0.640**	**0.642**	**0.620**	**0.666**

and enhances the model's generalization ability and adaptability. Finally, unlike the baseline approaches, VulTC-LTPF employs a bi-modal input strategy that combines information from source code and vulnerability descriptions. With the bi-modal input, VulTC-LTPF is able to capture the potential correlation between source code and vulnerability descriptions, addressing the limitations of single-modal input.

> **Summary for RQ1:** Our proposed approach, VulTC-LTPF outperforms the baselines across all evaluated metrics. Specifically, VulTC-LTPF achieves improvements in F1 score ranging from 26.3% to 49.8% and in MCC from 26.1% to 55.1%. These results underscore the effectiveness of VulTC-LTPF in addressing SVTC tasks.

5.2. RQ2: What is the impact of adaptive data augmentation on the performance of the VulTC-LTPF approach?

Approach: To demonstrate the effectiveness of the adaptive data augmentation component, we validate it through ablation experiments, with "w/o ADA" representing the approach without adaptive data augmentation. The rest of the experimental settings are kept constant. The comparison results are shown in Table 4.

Results: The results of the ablation experiments, as shown in Table 4, indicate that the ADA strategy has a significant effect on the model performance. When the ADA strategy is removed (i.e., "w/o ADA"), the model's accuracy is 0.654 (3.4% reduction), precision is 0.6 (4% reduction), recall is 0.615 (2.7% reduction), F1 score is 0.582 (3.8% reduction), and MCC is 0.629 (3.7% reduction). This shows that the ADA strategy effectively mitigates the problem of long-tailed distribution of the data set by enhancing the diversity and quality of the tail class samples, which improves the model's ability to detect vulnerabilities in a small number of classes and enhances the overall performance of the model.

To verify the effectiveness of adaptive data augmentation further, we designed additional experiments. We compared VulTC-LTPF with a version of the model that uses a traditional data augmentation strategy [55], where the adaptive mechanism is removed(i.e., "w/o Adaptive").

The experimental results are shown in Table 5. The results show that the adaptive data augmentation approach is better than the traditional data augmentation model in all evaluation metrics. This may be because the traditional data augmentation strategy uses a static approach to treat all tail classes equally and cannot be dynamically adjusted according to the actual performance of the categories during training. As a result, it is difficult for the model to pay extra attention to the poorly performing categories during training, limiting its learning ability on these categories and the final overall classification effect.

This confirms that the adaptive mechanism is crucial for better mitigating the long-tailed distribution problem and improving classification performance.

Table 6
Comparison results between our approaches VulTC-LTPF and VulTC-LTPF alone using different modal information as input, with the best results highlighted in bold.

Approach	Accuracy	Precision	Recall	F1	MCC
w/o Desc	0.182	0.215	0.152	0.146	0.118
w/o Code	0.641	0.606	0.581	0.560	0.616
VulTC-LTPF	**0.688**	**0.640**	**0.642**	**0.620**	**0.666**

> **Summary for RQ2:** The results of the ablation experiments indicate that the adoption of adaptive data augmentation leads to significant improvements across multiple metrics. This demonstrates the effectiveness of ADA in enabling the model to address the long-tailed distribution present in the dataset.

5.3. RQ3: What is the impact of bi-modal information on the performance of the VulTC-LTPF approach?

Approach: In order to demonstrate the effectiveness of the bi-modal information input, we verified it through ablation experiments, using "w/o Desc" to represent the approach that does not use the description as input and "w/o Code" to represent the approach that does not use the source code as input. The rest of the experimental settings are kept constant. The comparison results are shown in Table 6.

Results: The results of the ablation experiments are shown in Table 6. Specifically, when only the vulnerability description ("w/o Desc") is used as input, the performance of the model is significantly degraded, with a F1 score of 0.146, and an MCC of 0.118. This suggests that relying on source vulnerability information alone is not sufficient to effectively capture critical information about vulnerabilities. In contrast, when only the source code ("w/o Code") is used as input, the performance improves with a F1 score of 0.56, and MCC of 0.616. This suggests that the vulnerability description provides valuable information (e.g., contextual information and details that are not immediately apparent from the source code), but still falls short of optimal performance.

When source code and vulnerability descriptions are used together as input, there is a significant improvement in all metrics to achieve the best results. For example, the F1 score is 0.62 and the MCC is 0.666. These results show that by fusing source code and vulnerability descriptions, the input of bi-modal information can help the model capture richer vulnerability information. This information fusion approach provides a more comprehensive understanding of vulnerabilities, which significantly improves the model's performance on SVTC.

> **Summary for RQ3:** The model performance is relatively weak when using only source code or vulnerability descriptions, while the bi-modal inputs combining both significantly enhance the model performance.

5.4. RQ4: What is the impact of the prompt tuning paradigm on the performance of the VulTC-LTPF approach?

Approach: In order to bridge the gap between the pre-trained model and the downstream task, we introduce a specially tailored prompt tuning paradigm for the SVTC task. In this study, we validate the effectiveness of this paradigm by comparing the performance of VulTC-LTPF with traditional fine-tuning approaches. Specifically, traditional fine-tuning approaches focus only on tuning pre-trained models and do not involve task-based prompt design. In order to introduce the pre-training fine-tuning approach in bi-modal inputs, we include a special identifier `<desc>` before the vulnerability description to

Table 7
Comparison between VulTC-LTPF using the prompt tuning paradigm and VulTC-LTPF using the fine-tuning paradigm, with the best results highlighted in bold.

Approach	Accuracy	Precision	Recall	F1	MCC
VulTC-LTPF$_{ft}$	0.623	0.556	0.537	0.519	0.597
VulTC-LTPF	**0.688**	**0.640**	**0.642**	**0.620**	**0.666**

distinguish different parts of the input data. The input format is defined as follows:

$$X = X_{\text{code}} + \langle \text{desc} \rangle + X_{\text{desc}}$$

where X represents the overall input data, X_{code} is the source code part, and X_{desc} is the vulnerability description part. In this framework, the model using the traditional fine-tuning approach is represented as VulTC-LTPF$_{ft}$.

Results: The experimental results are shown in Table 7, and from the results in the table, it can be seen that the VulTC-LTPF approach significantly outperforms VulTC-LTPF$_{ft}$ in all the evaluation metrics. For example, VulTC-LTPF$_{ft}$ has an F1 score of 0.519 (10.1% reduction) and an MCC of 0.597 (6.9% reduction).

These results indicate that the prompt tuning approach is able to better utilize the latent knowledge of the pre-trained model, thus showing significant performance improvement in the SVTC task. Traditional fine-tuning approaches rely on large-scale annotated data for fine-tuning model parameters, however, this approach does not fully utilize the potential of the pre-trained model, resulting in lower efficiency and more limited performance when processing the task. Compared to the fine-tuning approach, prompt tuning is able to enhance the model's comprehension of vulnerability information more effectively by designing task-specific prompt templates that more accurately guide the knowledge of the pre-trained model to align with the requirements of the downstream tasks.

> **Summary for RQ4:** Compared with the fine-tuning approach, the prompt tuning paradigm significantly improves the performance of VulTC-LTPF.

5.5. RQ5: What is the impact of different prompt settings on VulTC-LTPF performance?

Approach: To validate the effectiveness of our proposed hybrid prompt templates, we designed comparison experiments comparing the hybrid prompt templates (i.e., Eq. (3)) with hard prompts only (i.e., Eq. (1)) and soft prompts only (i.e., Eq. (2)). The rest of the experimental settings were kept constant. The comparison results are shown in Table 8. In addition, to verify the effectiveness of the one-to-many verbalizer we used. We also set up a comparison experiment between the approaches of one-to-one and one-to-many verbalizers (as described by Eqs. (4) and (5), which have different numbers of mapping words). The results are shown in Table 9

Results: Tables 8 and 9 show the impact of different prompt settings and mapping words configurations on the performance of the VulTC-LTPF model. First, Table 8 shows a comparison between hard prompt, soft prompt and hybrid prompt templates. The results show that the F1 score and MCC are 0.6 and 0.63 when only hard prompt are used. The F1 score and MCC are 0.605 and 0.657 when only soft prompt are used. We found that the hybrid prompt template outperforms both hard prompt and soft prompt when used alone on all evaluation metrics.

The above results can be attributed to the fact that hybrid prompts can effectively combine the structured format of hard prompting with the flexibility of soft prompting. Hard prompting provides a clear structure that helps the model maintain consistency with pre-trained knowledge, while soft prompting increases the adaptability of the model and enables it to better capture the nuances in the data. Through this balance, hybrid prompting can more comprehensively capture vulnerability information, thereby achieving higher performance.

Table 8
Comparison of our approach VulTC-LTPF with approaches using different types of prompt templates. The best result is highlighted in bold.

Approach	Accuracy	Precision	Recall	F1	MCC
Hard	0.654	0.610	0.619	0.600	0.630
Soft	0.679	0.614	0.641	0.605	0.657
VulTC-LTPF	**0.688**	**0.640**	**0.642**	**0.620**	**0.666**

In addition, Table 9 shows the impact of different verbalizer configurations with different mapping words on the performance of the VulTC-LTPF model. In this experiment, the model performs best when using one-to-many verbalizer with two mapping words. In contrast, the MCC decreased by 3.8% when using a configuration of three mapping words, and the F1 score decreased by 8.4% and the MCC decreased by 5.4% when using a one-to-one verbalizer. Therefore, the configuration of two mapping words has achieved the best balance, which not only enriches the mapping representation, but also does not introduce irrelevant information, making it the most effective configuration.

> **Summary for RQ5:** Our proposed hybrid prompt template and one-to-many verbalizer achieve significant performance improvements over other settings. These results show that the performance of VulTC-LTPF can be effectively improved by optimizing the settings of prompt templates and verbalizers.

6. Discussion

6.1. Influence of the choice of scaling factor k

In the design of our adaptive data augmentation, the scaling factor k plays a crucial role in determining the number of augmented samples generated for each tail class based on its error rate. The value of k influences how aggressively the augmentation is applied to tail classes, which can impact both the model's performance and the class balance in the training dataset.

To ensure the robustness and reproducibility of our adaptive data augmentation, we experiment with the grid search approach [47] to select a suitable scaling factor k. Specifically, we tested values of $k \in \{1, 5, 10, 15, 20\}$, covering both conservative and aggressive augmentation strategies. These values range from conservative to aggressive augmentation strategies, reflecting different augmentation strengths. By experimentally comparing the model performance under each candidate value, we aim to determine an appropriate parameter value that can augment the tail classes while maintaining overall performance. The experimental results are shown in Table 10.

Our experimental results indicate that $k = 15$ yielded the best performance in terms of model accuracy and generalization. Specifically, this value of k produced a sufficient number of augmented samples to address the underrepresentation and misclassification of tail classes, without introducing excessive redundancy or noise into the training data. Although a higher k value (e.g., 20) may bring about further improvements in some tail classes, it also leads to overfitting and longer training times. In addition, a higher k value means that stronger augmentation is applied to the tail classes, which may lead to an excessive number of samples in some tail classes, thereby weakening the model's ability to learn from the head classes. This situation may trigger a decrease in the classification accuracy of the head classes, which will have a negative impact on the overall performance of the model. Conversely, smaller values of k (e.g., 5) were insufficient to alleviate the challenges posed by tail classes, resulting in suboptimal performance.

Table 9
Comparison of different verbalizers on VulTC-LTPF, with the best result highlighted in bold.

Verbalizer	Accuracy	Precision	Recall	F1	MCC
"CWE-119": ["buffer overflow"] "CWE-125": ["out-of-bounds read"] ..... "CWE-310": ["cryptographic issue"]	0.637	0.535	0.583	0.536	0.612
"CWE-119": [...+ "memory violation"] "CWE-125": [...+ "information leak"] ..... "CWE-310": [...+ "insecure cryptography"]	**0.688**	0.640	**0.642**	0.620	**0.666**
"CWE-119": [...+ "out-of-bounds access"] "CWE-125": [...+ "data exposure"] ..... "CWE-310": [...+ "weak encryption"]	0.652	**0.646**	0.637	**0.628**	0.628

Table 10
The effect of different scaling factor k values on the classification performance of VulTC-LTPF, with the best result highlighted in bold.

Value of k	Accuracy	Precision	Recall	F1	MCC
1	0.665	0.600	0.622	0.591	0.641
5	0.669	0.601	0.635	0.593	0.647
10	0.675	0.614	0.627	0.600	0.652
15	**0.688**	**0.640**	**0.642**	**0.620**	**0.666**
20	0.665	0.614	0.630	0.601	0.641

Table 11
The results of the VulTC-LTPF performance comparison when considering different PLMs are shown, with the best result for each performance metric highlighted in bold.

Approach	Accuracy	Precision	Recall	F1	MCC
T5	0.609	0.518	0.522	0.506	0.581
UnixCoder	0.644	0.548	0.570	0.545	0.619
CodeBERT	0.646	0.556	0.577	0.551	0.621
GraphCodeBERT	0.636	0.575	0.594	0.558	0.610
CodeT5	**0.688**	**0.640**	**0.642**	**0.620**	**0.666**

In summary, in the implementation of adaptive data enhancement, we designed a fixed k value and found a suitable k value through experiments. In the experiment, our main purpose is not to find the best parameters, but to prove the effectiveness of our approach. Our results suggest that $k = 15$ strikes a good balance between augmentation intensity and model efficiency, making it the most effective choice for improving the classification performance of tail classes in this study. In future work, we can further explore more adaptive dynamic scaling strategies, such as automatically adjusting parameters based on the distribution characteristics or training performance of samples in each category, so as to achieve more flexible, responsive and data-driven adaptive data augmentation.

6.2. Influence of PLMs

To comprehensively evaluate our proposed VulTC-LTPF approach, we conducted a comparison experiment using different pre-trained language models (PLMs) to assess their effectiveness in the SVTC task. Specifically, we selected five widely used PLMs for comparison: T5 [56], UnixCoder [57], CodeBERT [58], GraphCodeBERT [59], and CodeT5 [27]. To ensure a fair comparison, we only replaced the PLM and kept all other experimental settings the same. The results of this experiment are shown in Table 11.

The experimental results show that VulTC-LTPF using CodeT5 consistently outperforms those using other PLM models in all evaluation metrics. Compared with using T5, a general pre-trained model, the use of CodeT5 in the VulTC-LTPF achieved a significant improvement. T5 has a MCC of 0.581, which is relatively average. Since T5 is not specifically optimized for programming languages, it is difficult for it to effectively capture the source code syntax and semantic features required for the SVTC task, which affects its performance in the SVTC task. The MCC obtained by using UnixCoder's VulTC-LTPF is improved to 0.619, but it is still lower than that of using CodeT5. UnixCoder is based on the UniLM style of design, and may be subject to interference between tasks when processing multiple tasks, limiting its performance in specific tasks.

Although CodeBERT and GraphCodeBERT are models optimized for coding tasks, unlike CodeT5, they rely on the BERT architecture, which may limit their ability to fully capture the deep semantic relationship between source code and vulnerability descriptions.

The reason why CodeT5 outperforms other models in various metrics is due to its design specifically for code syntax and natural language understanding. Unlike other models, CodeT5 is optimized to better handle the bi-modal input (source code and vulnerability description) in the SVTC task, which makes CodeT5 the best choice for the VulTC-LTPF approach.

6.3. Influence of the design of the prompt template

As shown in Section 5.5, we conducted comparative experiments on soft prompt, hard prompt, and hybrid prompt templates to study the effectiveness of different types of prompt templates. The experimental results show that the hybrid prompt in VulTC-LTPF has the best performance. To further study the impact of different prompt designs on the performance of our approach, we refer to previous research [18] and explore it by designing different prompt templates.

Since soft prompt tokens are automatically optimized during model training, we only initialize them at the beginning of the experiment for reproducibility. Therefore, we focus on the hard prompt part in this experiment. We designed two other prompt templates by modifying the hard prompt parts. One of them uses longer hard prompt tokens, and the template is as follows:

$$f_{\text{Long}} = \text{The following is a source code that contains a security vulnerability: [X] The following is the description text of the vulnerability: [Y] [SOFT] [Z]} \quad (15)$$

The other uses shorter hard prompt tokens, and the template is as follows:

$$f_{Short} = \text{code: [X] description: [Y] [SOFT] [Z]} \quad (16)$$

We conducted comparative experiments based on these three different prompt template designs, and the detailed results are shown in Table 12.

From the experimental results, we can see that the design of different prompt templates does have a certain impact on the performance of VulTC-LTPF. In the experiment, we found that when using prompt templates with hard prompt tokens of appropriate length, better performance was obtained on all evaluation metrics. In contrast, the long prompt may introduce redundant information and distract the model's

Table 12
Comparison of our approach VulTC-LTPF with approaches using prompt templates with different prompt tokens. The best result is highlighted in bold.

Approach	Accuracy	Precision	Recall	F1	MCC
Long	0.668	0.629	0.637	0.612	0.645
Short	0.664	0.605	0.629	0.592	0.642
VulTC-LTPF	**0.688**	**0.640**	**0.642**	**0.620**	**0.666**

Table 13
Wilcoxon signed-rank test between VulTC-LTPF and the baselines on MCC.

Approach	p-values
$\text{VulExplainer}_{\text{CodeBERT}}$	**
$\text{VulExplainer}_{\text{CodeGPT}}$	**
Devign	**
ReGVD	**

Note: *** means p-value < 0.001, ** means p-value < 0.01, * means p-value < 0.005

attention, while the short prompt may lack sufficient guidance to help the model align with the task objective. These findings further emphasize the critical importance of the appropriate prompt design for the effectiveness of prompt tuning.

6.4. Statistical significance test

The Wilcoxon signed-rank test [60] is commonly used to assess whether there is a significant difference between two paired sample distributions. We performed the Wilcoxon signed-rank test to assess further the statistical significance of the improvement observed in our approach over the baselines.

Specifically, we ran our approach and each baseline model multiple times independently under the same experimental settings and recorded their corresponding MCC metrics. We then used the Wilcoxon signed-rank test to compare the score distributions of our approach and each baseline model to assess whether the differences between the two were statistically significant. The comparison results with the baselines are shown in Table 13.

The experimental results show that the p-values between VulTC-LTPF and all baselines are less than 0.01, indicating that there is a statistically significant difference in performance. This means that the performance improvement of our approach compared to the baselines is not due to random fluctuations, but has statistical support. The above results further verify the effectiveness and robustness of our approach.

6.5. Threats to validity

In this section, we discuss potential threats to the validity of our study.

Internal Validity. This threat is mainly related to the implementation of the VulTC-LTPF and baseline approaches. To mitigate this threat, we conduct a detailed code review and thorough testing of the approach implementation. In addition, to avoid the impact of configuration differences on baseline performance, we conduct experiments according to the recommended hyperparameter settings of these baselines. Although the hyperparameters of the model optimizer, learning rate, etc., as well as the design of prompt templates and verbalizers, may still affect the results, it is more difficult to comprehensively optimize these factors. However, the experimental results show that under the current configuration, the performance of the VulTC-LTPF approach is always better than that of the baseline and the traditional fine-tuning paradigm, thus verifying its effectiveness.

External Validity. This threat is mainly related to the choice of dataset. First, we selected the high-quality MegaVul dataset as the initial dataset, which covers a wide range of time (from 2006 to 2024), has a wealth of vulnerability types and real-world code scenarios, and is highly representative. To further improve its adaptability, we have optimized the dataset, such as cleaning up redundant data and updating deprecated CWE-IDs, to serve the SVTC task better. Second, the current experiment only uses the C/C++ code dataset, which may limit the generality of the approach. However, the core design of VulTC-LTPF does not rely on a specific programming language. In theory, it can be extended to datasets of other languages to support a broader range of vulnerability classification tasks. In addition, the potential drawbacks of the hybrid prompt tuning process or scalability concerns may pose a threat to the validity of the approach. Since the current design relies on manually crafted templates and verbalizer mappings, applying the approach to other domains or tasks may require substantial human effort. Future work could address these issues by introducing automated or semi-automated prompt engineering approaches.

Construct Validity. This threat is mainly related to the selection of evaluation metrics. To mitigate this threat, we use multiple commonly used and reliable evaluation metrics to comprehensively evaluate the performance of VulTC-LTPF and the baseline approaches.

7. Related work

Software vulnerability type classification is a key step in the software vulnerability repair process. It can effectively help developers quickly identify the type of vulnerability, providing an important basis for subsequent vulnerability repair. With the rapid increase in the number of vulnerabilities, the traditional manual classification approach has difficulty coping with the processing requirements of a large number of vulnerabilities. Therefore, there is an urgent need to design an efficient automated SVTC approach to improve the accuracy and efficiency of vulnerability classification. From the perspective of vulnerability information input, current SVTC research can be broadly divided into two categories: approaches that utilize vulnerability descriptions [13,14,21] and approaches based on source code analysis [12,15,19,20]. These approaches have different perspectives and use different modalities of vulnerability information to propose their own solutions to the problem of SVTC. In addition, since the distribution of vulnerability types in the dataset often shows a long-tailed distribution, the lack of a small number of tail class samples makes the classification effect of the model worse in these classes. Therefore, in recent years, some studies [12,16,17] have begun to try to introduce long-tailed learning approaches to improve the model's ability to identify tail classes, and thus improve the overall performance of the model in the SVTC task.

Vulnerability Description-Based Approaches Vulnerability descriptions provide a textual description of software vulnerabilities, usually detailing their characteristics, potential consequences and exploitation scenarios. This textual data has been extensively studied for the purpose of automating SVTC tasks [21]. Aota et al. [13] used machine learning models to classify vulnerability types based on the textual features of vulnerability descriptions. Pan et al. [14] adopted a deep learning approach combining BiGRU and TextCNN models, which effectively captured the sequential features and contextual information in vulnerability descriptions, improving the accuracy of vulnerability classification.

Source Code-Based Approaches In addition to textual descriptions, analyzing source code has become another important approach in SVTC [19,20]. Code-level analysis provides fine-grained and detailed information, capturing structural and contextual aspects of vulnerabilities. For example, Fu et al. [15] open by introducing a hierarchical distillation mechanism that uses vulnerability code information to provide higher classification transparency and more accurate vulnerability type classification . Ji et al. [12] explore the application of the contrastive learning approach in code vulnerability types classification, using a contrastive learning strategy to enhance the classification ability of the model through an efficient representation of code features.

Long-tailed Learning-Based Approaches The long-tailed learning approach offers an effective solution to address the imbalanced distribution of vulnerability types within software vulnerability datasets

[16]. Wen et al. [17] explored the loss function re-weighting approach in long-tailed learning and proposed the LIVABLE framework. This framework addresses the long-tailed problem by designing an adaptive re-weighting module, which dynamically adjusts the category weights based on the samples size and the training progress. By ensuring that the tail categories receive sufficient attention during training, this dynamic weighting strategy significantly improves the model's performance on SVTC.

Compared with existing work, our approach focuses on the integration bi-modal information from source code and vulnerability descriptions. By leveraging prompt tuning, we effectively fuse this bi-modal information to improve the performance of SVTC. Furthermore, in addressing the challenges posed by long-tailed learning, we introduce adaptive data augmentation informed by a comprehensive analysis of existing data augmentation approaches. This strategy successfully alleviates the long-tailed distribution problem and improves model performance.

8. Conclusion

We propose a novel SVTC approach, refer to as VulTC-LTPF, which combines source code and vulnerability descriptions to accurately predict vulnerability types. To address the long-tailed distribution of vulnerability types within datasets, we introduce adaptive data augmentation to improve the model's learning ability for tail classes. Additionally, the performance of the model is further improved through the optimization of the prompt tuning paradigm. To achieve a more comprehensive representation of vulnerabilities, VulTC-LTPF employs bi-modal information fusion, combining the structured features of source code with the textual information of vulnerability descriptions. Experimental results show that VulTC-LTPF outperforms existing baseline approaches in terms of accuracy, precision, recall, F1 score and MCC metrics. These findings validate the superiority of VulTC-LTPF in the SVTC task.

For future work, we plan to explore more sophisticated augmentation techniques, such as code refactoring and syntax transformations, to further enhance the diversity of training data. Additionally, we aim to explore a dynamic scaling strategy that can automatically adjust parameters further to enhance the flexibility and adaptability of adaptive data augmentation. Lastly, we plan to develop more effective prompt configurations specifically tailored for SVTC to improve model performance continuously. We also intend to explore automated or semi-automated prompt optimization techniques to reduce manual effort and enhance the scalability of the approach, thereby further improving the prompt tuning module.

CRediT authorship contribution statement

Long Zhang: Writing – original draft, Visualization, Validation, Software, Methodology. **Xiaolin Ju:** Writing – review & editing, Supervision, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Lina Gong:** Writing – review & editing, Supervision, Methodology, Investigation, Conceptualization. **Jiyu Wang:** Writing – review & editing, Software, Resources, Data curation. **Zilong Ren:** Validation, Software, Methodology, Investigation, Formal analysis, Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

References

[1] V. Smyth, Software vulnerability management: how intelligence helps reduce the risk, Netw. Secur. 2017 (3) (2017) 10–12.

[2] C. Zhang, H. Liu, J. Zeng, K. Yang, Y. Li, H. Li, Prompt-enhanced software vulnerability detection using chatgpt, in: Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, 2024, pp. 276–277.

[3] R.L. Alaoui, E.H. Nfaoui, Deep learning for vulnerability and attack detection on web applications: A systematic literature review, Futur. Internet 14 (4) (2022) 118.

[4] CWE top 25 most dangerous software weaknesses, 2024, Accessed: 2025 https://cwe.mitre.org/top25/archive/2024/2024_cwe_top25.html.

[5] CWE-787: out-of-bounds write, 2025, Accessed: 2025 https://cwe.mitre.org/data/definitions/787.html.

[6] Y. Liu, C. Wang, Y. Ma, DL4SC: a novel deep learning-based vulnerability detection framework for smart contracts, Autom. Softw. Eng. 31 (1) (2024) 24.

[7] C. Liu, X. Chen, X. Li, Y. Xue, Making vulnerability prediction more practical: Prediction, categorization, and localization, Inf. Softw. Technol. 171 (2024) 107458.

[8] G. Lu, X. Ju, X. Chen, W. Pei, Z. Cai, GRACE: Empowering LLM-based software vulnerability detection with graph structure and in-context learning, J. Syst. Softw. 212 (2024) 112031.

[9] Z. Cai, Y. Cai, X. Chen, G. Lu, W. Pei, J. Zhao, CSVD-TF: Cross-project software vulnerability detection with TrAdaBoost by fusing expert metrics and semantic metrics, J. Syst. Softw. 213 (2024) 112038.

[10] Common vulnerability scoring system SIG, 2025, Accessed: 2025 https://www.first.org/cvss.

[11] CWE-22: improper limitation of a pathname to a restricted directory ('path traversal'), 2025, Accessed: 2025 https://cwe.mitre.org/data/definitions/22.html.

[12] C. Ji, S. Yang, H. Sun, Y. Zhang, Applying contrastive learning to code vulnerability type classification, in: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, 2024, pp. 11942–11952.

[13] M. Aota, H. Kanehara, M. Kubo, N. Murata, B. Sun, T. Takahashi, Automation of vulnerability classification from its description using machine learning, in: 2020 IEEE Symposium on Computers and Communications, ISCC, IEEE, 2020, pp. 1–7.

[14] M. Pan, P. Wu, Y. Zou, C. Ruan, T. Zhang, An automatic vulnerability classification framework based on BiGRU-TextCNN, Procedia Comput. Sci. 222 (2023) 377–386.

[15] M. Fu, V. Nguyen, C.K. Tantithamthavorn, T. Le, D. Phung, Vulexplainer: A transformer-based hierarchical distillation for explaining vulnerability types, IEEE Trans. Softw. Eng. (2023).

[16] X. Deng, F. Duan, R. Xie, W. Ye, S. Zhang, Improving long-tail vulnerability detection through data augmentation based on large language models, in: 2024 IEEE International Conference on Software Maintenance and Evolution, ICSME, IEEE, 2024, pp. 262–274.

[17] X.-C. Wen, C. Gao, F. Luo, H. Wang, G. Li, Q. Liao, LIVABLE: exploring long-tailed classification of software vulnerability types, IEEE Trans. Softw. Eng. (2024).

[18] C. Wang, Y. Yang, C. Gao, Y. Peng, H. Zhang, M.R. Lyu, Prompt tuning in code intelligence: An experimental evaluation, IEEE Trans. Softw. Eng. 49 (11) (2023) 4869–4885.

[19] M. Fu, C.K. Tantithamthavorn, V. Nguyen, T. Le, Chatgpt for vulnerability detection, classification, and repair: How far are we? in: 2023 30th Asia-Pacific Software Engineering Conference, APSEC, IEEE, 2023, pp. 632–636.

[20] G. Lu, X. Ju, X. Chen, S. Yang, L. Chen, H. Shen, Assessing the effectiveness of vulnerability detection via prompt tuning: An empirical study, in: 2023 30th Asia-Pacific Software Engineering Conference, APSEC, IEEE, 2023, pp. 415–424.

[21] Q. Wang, Y. Gao, J. Ren, B. Zhang, An automatic classification algorithm for software vulnerability based on weighted word vector and fusion neural network, Comput. Secur. 126 (2023) 103070.

[22] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, G. Neubig, Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing, ACM Comput. Surv. 55 (9) (2023) 1–35.

[23] L. Yu, J. Lu, X. Liu, L. Yang, F. Zhang, J. Ma, PSCVFinder: A prompt-tuning based framework for smart contract vulnerability detection, in: 2023 IEEE 34th International Symposium on Software Reliability Engineering, ISSRE, IEEE, 2023, pp. 556–567.

[24] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, R. Feris, Spottune: transfer learning through adaptive fine-tuning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 4805–4814.

[25] X.L. Li, P. Liang, Prefix-tuning: Optimizing continuous prompts for generation, in: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), 2021, pp. 4582–4597.

[26] Z. Shen, Z. Liu, J. Qin, M. Savvides, K.-T. Cheng, Partial is better than all: Revisiting fine-tuning strategy for few-shot learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35, 2021, pp. 9594–9602.

[27] Y. Wang, W. Wang, S. Joty, S.C. Hoi, CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation, in: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, 2021, pp. 8696–8708.
[28] Y. Zhou, S. Liu, J. Siow, X. Du, Y. Liu, Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks, Adv. Neural Inf. Process. Syst. 32 (2019).
[29] V.-A. Nguyen, D.Q. Nguyen, V. Nguyen, T. Le, Q.H. Tran, D. Phung, ReGVD: Revisiting graph neural networks for vulnerability detection, in: Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings, 2022, pp. 178–182.
[30] Common vulnerabilities and exposures, 2024, Accessed: 2025 https://cwe.mitre.org.
[31] Y. Zhang, B. Kang, B. Hooi, S. Yan, J. Feng, Deep long-tailed learning: A survey, IEEE Trans. Pattern Anal. Mach. Intell. 45 (9) (2023) 10795–10816.
[32] J. Tan, C. Wang, B. Li, Q. Li, W. Ouyang, C. Yin, J. Yan, Equalization loss for long-tailed object recognition, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 11662–11671.
[33] L. Yang, H. Jiang, Q. Song, J. Guo, A survey on long-tailed visual recognition, Int. J. Comput. Vis. 130 (7) (2022) 1837–1872.
[34] J.-X. Shi, T. Wei, Y. Xiang, Y.-F. Li, How re-sampling helps for long-tail learning? Adv. Neural Inf. Process. Syst. 36 (2023).
[35] H. Peng, W. Pian, M. Sun, P. Li, Dynamic re-weighting for long-tailed semi-supervised learning, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2023, pp. 6464–6474.
[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., Imagenet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (2015) 211–252.
[37] C. Wei, S.M. Xie, T. Ma, Why do pretrained language models help in downstream tasks? an analysis of head and prompt tuning, Adv. Neural Inf. Process. Syst. 34 (2021) 16158–16170.
[38] X. Han, W. Zhao, N. Ding, Z. Liu, M. Sun, Ptr: Prompt tuning with rules for text classification, AI Open 3 (2022) 182–192.
[39] C. Wang, Y. Yang, C. Gao, Y. Peng, H. Zhang, M.R. Lyu, No more fine-tuning? an experimental evaluation of prompt tuning in code intelligence, in: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022, pp. 382–394.
[40] S. Yang, X. Chen, K. Liu, G. Yang, C. Yu, Automatic bi-modal question title generation for stack overflow with prompt learning, Empir. Softw. Eng. 29 (3) (2024) 63.
[41] C. Ni, L. Shen, X. Yang, Y. Zhu, S. Wang, MegaVul: AC/C++ vulnerability dataset with comprehensive code representations, in: 2024 IEEE/ACM 21st International Conference on Mining Software Repositories, MSR, IEEE, 2024, pp. 738–742.
[42] Z. Ren, X. Ju, X. Chen, H. Shen, ProRLearn: boosting prompt tuning-based vulnerability detection by reinforcement learning, Autom. Softw. Eng. 31 (2) (2024) 38.
[43] M. Tsimpoukelli, J.L. Menick, S. Cabi, S. Eslami, O. Vinyals, F. Hill, Multimodal few-shot learning with frozen language models, Adv. Neural Inf. Process. Syst. 34 (2021) 200–212.
[44] Z. Zhang, M. Sabuncu, Generalized cross entropy loss for training deep neural networks with noisy labels, Adv. Neural Inf. Process. Syst. 31 (2018).
[45] X. Ruan, Y. Yu, W. Ma, B. Cai, Prompt learning for developing software exploits, in: Proceedings of the 14th Asia-Pacific Symposium on Internetware, 2023, pp. 154–164.
[46] C. Niu, C. Li, V. Ng, D. Chen, J. Ge, B. Luo, An empirical comparison of pre-trained models of source code, in: 2023 IEEE/ACM 45th International Conference on Software Engineering, ICSE, IEEE, 2023, pp. 2136–2148.
[47] I. Syarif, A. Prugel-Bennett, G. Wills, SVM parameter optimization using grid search and genetic algorithm to improve classification performance, TELKOMNIKA (Telecommun. Comput. Electron. Control) 14 (4) (2016) 1502–1509.
[48] J. Fan, Y. Li, S. Wang, T.N. Nguyen, AC/C++ code vulnerability dataset with code changes and CVE summaries, in: Proceedings of the 17th International Conference on Mining Software Repositories, 2020, pp. 508–512.
[49] Category ID: 17. deprecated, 2025, Accessed: 2025 https://cwe.mitre.org/data/definitions/17.html.
[50] Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel, Gated graph sequence neural networks, 2015, arXiv preprint arXiv:1511.05493.
[51] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, 2016, arXiv preprint arXiv:1609.02907.
[52] N. Ding, S. Hu, W. Zhao, Y. Chen, Z. Liu, H. Zheng, M. Sun, OpenPrompt: An open-source framework for prompt-learning, in: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, 2022, pp. 105–113.
[53] D.P. Kingma, J.L. Ba, Adam: A method for stochastic optimization, 2015.
[54] L. Prechelt, Early stopping-but when? in: Neural Networks: Tricks of the Trade, Springer, Berlin, Heidelberg, 2002, pp. 55–69.
[55] C. Shorten, T.M. Khoshgoftaar, B. Furht, Text data augmentation for deep learning, J. Big Data 8 (1) (2021) 101.
[56] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P.J. Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, J. Mach. Learn. Res. 21 (140) (2020) 1–67.
[57] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, J. Yin, Unixcoder: Unified cross-modal pre-training for code representation, 2022, arXiv preprint arXiv:2203.03850.
[58] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, et al., Codebert: A pre-trained model for programming and natural languages, 2020, arXiv preprint arXiv:2002.08155.
[59] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, et al., Graphcodebert: Pre-training code representations with data flow, 2020, arXiv preprint arXiv:2009.08366.
[60] R.F. Woolson, Wilcoxon signed-rank test, Encycl. Biostat. 8 (2005).