VulExplainer: A Transformer-Based Hierarchical Distillation for Explaining Vulnerability Types

Michael Fu, *Student Member, IEEE*, Van Nguyen, *Member, IEEE*, Chakkrit (Kla) Tantithamthavorn, *Member, IEEE*, Trung Le, *Member, IEEE*, and Dinh Phung, *Member, IEEE*

***Abstract*—Deep learning-based vulnerability prediction approaches are proposed to help under-resourced security practitioners to detect vulnerable functions. However, security practitioners still do not know what type of vulnerabilities correspond to a given prediction (aka CWE-ID). Thus, a novel approach to explain the type of vulnerabilities for a given prediction is imperative. In this paper, we propose *VulExplainer*, an approach to explain the type of vulnerabilities. We represent *VulExplainer* as a vulnerability classification task. However, vulnerabilities have diverse characteristics (i.e., CWE-IDs) and the number of labeled samples in each CWE-ID is highly imbalanced (known as a highly imbalanced multi-class classification problem), which often lead to inaccurate predictions. Thus, we introduce a Transformer-based hierarchical distillation for software vulnerability classification in order to address the highly imbalanced types of software vulnerabilities. Specifically, we split a complex label distribution into sub-distributions based on CWE abstract types (i.e., categorizations that group similar CWE-IDs). Thus, similar CWE-IDs can be grouped and each group will have a more balanced label distribution. We learn TextCNN teachers on each of the simplified distributions respectively, however, they only perform well in their group. Thus, we build a transformer student model to generalize the performance of TextCNN teachers through our hierarchical knowledge distillation framework. Through an extensive evaluation using the real-world 8,636 vulnerabilities, our approach outperforms all of the baselines by 5%–29%. The results also demonstrate that our approach can be applied to Transformer-based architectures such as CodeBERT, GraphCodeBERT, and CodeGPT. Moreover, our method maintains compatibility with any Transformer-based model without requiring any architectural modifications but only adds a special distillation token to the input. These results highlight our significant contributions towards the fundamental and practical problem of explaining software vulnerability.**

***Index Terms*—Software vulnerability, software security.**

I. Introduction

As the number of discovered software vulnerabilities hit an all-time high of 20k in 2021 reported by National Vulnerability Database (NVD) [54], large software companies are spending more and more funds mitigating the security threats by granting bug bounties [24], [27], [49]. Software vulnerabilities are system weaknesses and glitches that can be further exploited by attackers to steal sensitive data or spread ransomware. Back in 2000, NVD has been created by the U.S. government to analyze and track new vulnerabilities to mitigate software security breaches. Another community-developed Common Weakness Enumeration (CWE) [13] list consists of multiple CWE-IDs representing various categories of vulnerability, where some CWE-IDs are easier to be exploited than others, hence requiring higher priority to be resolved. For instance, the widespread Log4j flaw inside an open-source Java library provided by the Apache Software Foundation was found at the end of 2021. Such flaw includes different CWE-IDs such as CWE-20 (i.e., improper input validation) and CWE-89 (i.e., improper neutralization of special elements used in an SQL command) with a high likelihood of exploitation [82]. Thus, it is important to recognize the type of vulnerability for a vulnerable program that enables security engineers to prioritize accordingly to focus on the more severe ones.

Various Deep Learning-based software vulnerability prediction (SVP) methods have been proposed that can even detect the vulnerabilities down to line-level [21], [28]. Nevertheless, those models can not identify what type of vulnerability is detected. The vulnerability type (i.e., CWE-ID) further explains the detected vulnerable code and helps security engineers understand and categorize the detected vulnerability to propose repairs or mitigation. Thus, software vulnerability classification (SVC) is an important task that supports SVP models by providing more explanation of detected vulnerable code, which could assist end users to comprehend the detected vulnerabilities. Recently, several automated SVC approaches have been proposed to identify the CWE-IDs given a vulnerable program or a vulnerability description using Machine Learning/Deep Learning models. In particular, transformer-based models were leveraged to achieve superior performance through the self-attention mechanism [18], [71]. However, due to the complexity and the nature of the process to collect and label software vulnerabilities wherein some popular vulnerabilities are highly reported while other unpopular ones are rarely reported, the distribution of different software vulnerabilities is highly imbalanced with some highly and rarely occurring CWE-IDs in real-world datasets. For instance, CWE-119 is a common buffer overflow vulnerability that has 2,127 samples in our experimental dataset, while CWE-94 is a more specific vulnerability about

Manuscript received 7 May 2023; revised 10 August 2023; accepted 11 August 2023. Date of publication 16 August 2023; date of current version 17 October 2023. Chakkrit (Kla) Tantithamthavorn was supported in part by the ARC's DECRA Fellowship (DE200100941). Recommended for acceptance by D. Hao. *(Corresponding author: Chakkrit (Kla) Tantithamthavorn.)*

The authors are with the Faculty of Information Technology, Monash University, Melbourne, Australia (e-mail: yeh.fu@monash.edu; van.nguyen1@monash.edu; chakkrit@monash.edu; trunglm@monash.edu; dinh.phung@monash.edu).

Digital Object Identifier 10.1109/TSE.2023.3305244

Code Injection that only has 11 samples. Such an imbalanced nature of CWE-IDs leads to a long-tailed label distribution that hinders the learning process of deep learning and transformer-based models, where models could learn too well on the specific CWE-IDs while performing poorly on other CWE-IDs.

Learning from a long-tailed label distribution has been widely studied in computer vision [8], [16], [44], [47], notably Focal Loss [44] and Logit Adjustment [47] methods. Although those methods have been demonstrated to couple well with CNNs and vision data, their direct application to transformer-based SVC does not perform satisfactorily. As shown in Table III, focal loss and logit adjustment do not improve transformer-based SVC in most cases. Additionally, some recent works have proposed to group data by label frequencies and use a balanced group softmax [41] or distillation [76] to learn a better model inspired by knowledge distillation [29] that enables transferring the knowledge from one or more teacher models to a student model. Again, although these approaches work to some extent for vision data and CNNs, they cannot improve transformer-based long-tailed SVC as shown in Table II (see the results for BAGS and LFME). We conjecture that grouping by label frequencies helps to mitigate the imbalance in each group. This operation in return creates groups of less similar CWE-IDs, hence making it harder to train a good teacher model for each group.

The goal of this work is to explain the type of vulnerabilities for detected vulnerable functions by classifying CWE-IDs. Thus, we need to address the aforementioned long-tailed label distribution that occurs in the SVC problem. To this end, we propose a hierarchical distillation approach based on the characteristics of vulnerabilities. Specifically, the CWE community has developed hierarchical CWE abstract types [14] to organize complex and diverse CWE-IDs by grouping similar CWE-IDs based on their characteristics. In practice, such categorization is more readable and understandable for security analysts. Moreover, each CWE abstract type becomes a more balanced distribution consisting of similar CWE-IDs as shown in Fig. 2, which enables us to learn a better model. Based on this observation, we propose a novel hierarchical distillation approach that is based on the hierarchical grouping of CWE-IDs to overcome the highly imbalanced problem. Particularly, we split a long-tailed label distribution Y into multiple distributions where each distribution corresponds to a specific CWE abstract type (i.e., Y_{Base}, $Y_{Category}$, Y_{Class}, $Y_{Variant}$, or $Y_{Deprecated}$) as depicted in Fig. 4. Our grouping strategy leads to multiple more balanced label distributions that consist of CWE-IDs with similar characteristics in each group, hence they are simpler for a DL model to learn from. Therefore, for each group corresponding to a CWE abstract type, we train a TextCNN teacher [37] to predict the CWE-IDs in this CWE abstract type. Additionally, to save up the computation and enable the training of the teachers simultaneously, we tie the backbone of the teachers, hence the teachers are only different in the classification heads for predicting the CWE-IDs belonging to their CWE abstract type. Finally, we invoke a transformer-based student to distill from multiple teachers, allowing it to generalize to the entire label distribution. Note that the idea of distilling a transformer from a different CNN teacher has been realized in the DeIT approach [68] for vision data. However, in our approach, we hierarchically distill from multiple TextCNN teachers based on the hierarchy of source code data.

Through an extensive evaluation of our VULEXPLAINER using the Big-Vul dataset [19] consisting of 3,754 vulnerabilities from 348 large-scale open-source software projects spanning from 2002 to 2019, we address the following three research questions:

- **(RQ1) What is the accuracy of our VULEXPLAINER for classifying software vulnerabilities (i.e., CWE-IDs)?**
  Results. Our VULEXPLAINER method achieves an accuracy of 65%–66% when applying to different transformer-based models, i.e., GraphCodeBERT [26], CodeBERT [20], and CodeGPT [46], which is 5%–29% more accurate than other baseline approaches.
- **(RQ2) Does VULEXPLAINER approach outperform loss-based methods for imbalanced data?**
  Results. Our approach outperforms the two loss-based methods and achieves the best performance for GraphCodeBERT, CodeBERT, and CodeGPT models.
- **(RQ3) What is the contribution of the components of our VULEXPLAINER?**
  Results. The ablation study reveals that our hierarchical grouping strategy achieves better performance than the grouping strategy that only focuses on label frequency. Furthermore, our TextCNN teacher models achieve advanced performance while being more efficient (requiring fewer parameters) than transformer-based teachers. Last but not least, the soft distillation (our method) that distills soft knowledge (probability distributions) is better than hard distillation that distills the hard predictions (one-hot predictions) of teacher models.

Novelty & Contributions. To the best of our knowledge, the contributions of this paper are as follows:

- VULEXPLAINER, a hierarchical software vulnerability distillation approach including two phases aiming to address the imbalanced data issue of SVC: (i) a novel data division approach to split a label distribution into multiple more balanced sub-distributions consisting of more similar CWE-IDs based on the hierarchical nature of CWE-IDs; (ii) a distillation approach based on the self-attention mechanism of transformer models to hierarchically distill knowledge from multiple TextCNN teachers based on the hierarchy of source code data.
- An extensive evaluation by comparing our VULEXPLAINER with seven competitive baseline approaches mentioned in Section IV-B.
- An empirical evaluation by comparing our VULEXPLAINER with two advanced loss-based methods (i.e., focal loss and logit adjustment) proposed for the imbalanced data issue.
- A complete ablation study to investigate each step of our VULEXPLAINER approach.

Paper Organization. Section II describes the background and problem statement. Section III presents our VULEXPLAINER framework. Section IV presents the experimental setup and results. Section V presents an additional discussion. Section VI

presents the related works. Section VII discloses the threats to validity. Section VIII draws the conclusions.

II. Background & Problem Statement

A. Background

Common Weakness Enumeration (CWE) is a community-developed list of software and hardware weaknesses and vulnerabilities. CWE provides a hierarchical categorization of software vulnerabilities where each CWE-ID determines a vulnerability type and each CWE abstract type determines a group of similar vulnerability types. Such a hierarchical categorization serves as a common language of software vulnerabilities, that helps security analysts identify and understand security flaws existing in software.

Recently, line-level software vulnerability prediction (SVP) methods [21], [28], [51], [57], [73] are proposed to detect vulnerable lines in source code that may save security analysts' efforts to locate vulnerabilities among a large number of codes. For instance, in Fig. 1, line-level SVP methods can detect the 9th line as a vulnerable line. However, SVP models can not provide further information such as CWE-IDs to explain the detected vulnerabilities. Therefore, such a lack of explainability concerns could hinder the adoption of SVP methods and make security analysts not fully understand the detected vulnerabilities, leading to more time spent on the security inspection [10], [32], [33], [36], [45], [56], [60], [64], [65], [66].

Therefore, software vulnerability classification (SVC) methods are proposed to classify vulnerable code into different CWE-IDs and explain the detected vulnerability [72] as shown at the bottom of Fig. 1. With the explanation provided by SVC methods, security analysts can understand more about vulnerability prediction by SVP models and efficiently suggest corresponding repair or mitigation strategies.

However, existing SVC methods still encounter an unresolved data imbalance issue. For instance, Das et al. [18] leveraged data augmentation [74] in their experiments but it did not further improve the performance of their transformer model. Wang et al. [72] also experienced the data imbalance issue and only focused on the top 10 frequency CWE-IDs in their experiment to mitigate the data imbalance, which hinders the model to identify rare vulnerability types. Our experimental dataset is also imbalanced; the samples per class range from 2127 to 10. Some approaches such as logit adjustment [47] and focal loss [44] were proposed in the vision domain to help DL models combat imbalanced label distribution for image classification tasks. Nevertheless, those methods from the vision domain have limited effect on improving transformer models for the SVC task as shown in Table III.

To combat the imbalanced label distribution, we propose to simplify one complex/imbalanced data distribution into multiple simple/balanced data distributions based on the hierarchical characteristics of software vulnerability data (i.e., CWE abstract types) mentioned early this section. We then leverage a teacher-student knowledge distillation method to benefit from the divided balance distributions as detailed in Section III. The studied data set used in this paper was crawled from the CVE

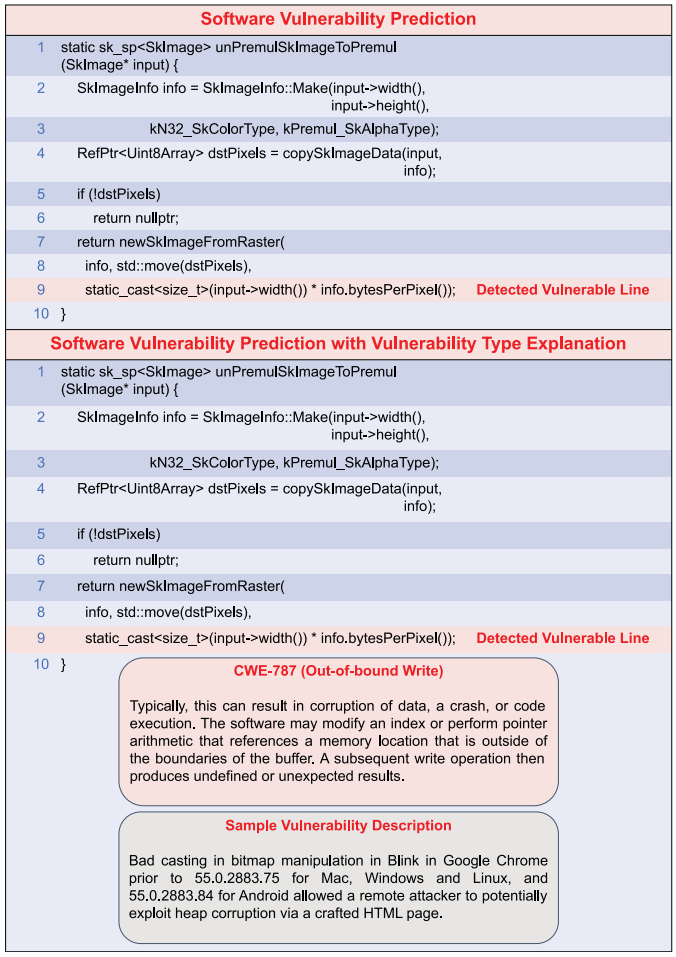

Fig. 1. A real-world vulnerability example of CWE-787 [1]. The upper part shows the vulnerability prediction generated by line-level SVP models while the lower part presents the same prediction with an extended explanation provided by the SVC approaches to illustrate the detected vulnerability.

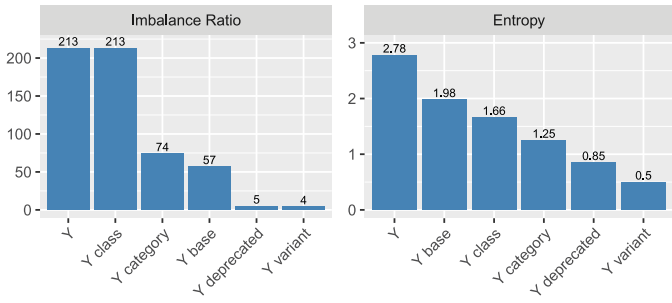

Fig. 2. Statistics that measure the imbalance of grouped and ungrouped data distributions.

database by Fan et al. [19] where the information of CWE-IDs and CWE abstract types have been labeled by human experts. Below, we introduce CWE abstract types, followed by an overview of the teacher-student knowledge distillation method.

1) CWE Abstract Types: CWE abstract types can be assessed through five dimensions that help characterize weaknesses within the Common Weakness Enumeration (CWE) system. These dimensions include behavior, which refers to observable actions or patterns associated with weakness. The

property focuses on specific attributes or qualities related to weaknesses. The technology identifies weaknesses that are specific to certain technologies or platforms. Language pertains to weaknesses that are specific to programming languages. Finally, resource relates to weaknesses that impact system resources.

In this paper, we consider five common CWE abstract types, namely Class, Base, Category, Variant, and Deprecated, each providing valuable insights into different aspects of weaknesses within the Common Weakness Enumeration (CWE) system. The class represents weaknesses described in a highly abstract manner, devoid of specific language or technology references. These weaknesses are typically characterized by 1 or 2 dimensions, including behavior, property, and resource. Base weaknesses, on the other hand, are described in an abstract fashion, but with sufficient details to infer specific detection and prevention methods. They offer a level of specificity between Class and Variant weaknesses. Base-level weaknesses encompass 2 or 3 dimensions, incorporating aspects such as behavior, property, technology, language, and resource. Category serves as a structural element that aids users in identifying weaknesses that share common characteristics, facilitating efficient grouping and classification. Deprecate encompasses all deprecated CWE-IDs. Lastly, Variant weaknesses are linked to specific types of products, often associated with particular languages or technologies. These weaknesses offer a higher level of specificity compared to Base weaknesses and are described in terms of 3 to 5 dimensions, encompassing behavior, property, technology, language, and resource. We refer interested readers to the official CWE documentation [14] for concrete examples of these abstract types.

2) Knowledge Distillation: By grouping similar CWE-IDs based on CWE abstract types, similar CWE-IDs are grouped and the data becomes more balanced. Consequently, it becomes possible to train a collection of more precise CWE-ID classification models, with each model dedicated to a specific CWE abstract type. However, their scope is limited to identifying CWE-IDs within their respective abstract types for which they were trained. Consequently, to extend the performance of these models to cover all CWE-IDs across various abstract types, we employ knowledge distillation to construct a student model.

Knowledge distillation is a procedure wherein knowledge is transferred from a single model or a set of models, often referred to as teacher models, to a single model known as the student model. The student model leverages the collective knowledge from the specialized models, enabling it to generalize and provide an accurate classification for CWE-IDs across different abstract types. Knowledge distillation can be seen as a type of model compression technique, originally introduced by Bucilua [6].

In particular, we leverage response-based knowledge that focuses on the final output layer of the teacher model. The underlying assumption is that the student model will acquire the capability to emulate the predictions made by the teacher model. To achieve this, we employ a distillation loss function that measures the disparity between the logits of the student and teacher models. By minimizing this loss during training, the student model gradually improves its ability to make predictions that align with those of the teacher model. We illustrate more technical details in Section III-C.

Subsequently, we present an additional challenge encountered in the CWE-ID classification task, supported by a preliminary analysis. We then proceed to outline our proposed approach for mitigating this challenge.

B. Challenge & Motivation

Prior studies have proposed methods for automatically classifying CWE-IDs based on textual vulnerability descriptions [3], [50], [63]. These approaches typically involve models learning to recognize CWE-IDs based on keywords acquired during training. However, our objective is to assist software developers in identifying CWE-IDs at the early stages of software development, where vulnerability descriptions may not be available. Consequently, we aim to develop models that can make predictions solely based on the source code input, which poses a more challenging task. This is because the same keywords, such as variable or function names, can be associated with different CWE-IDs depending on the context.

Large language models (LLMs) such as ChatGPT [55] and BARD [25] have demonstrated their ability to conduct code analysis and generate boilerplate code. However, we have reservations regarding the effectiveness of large language models (LLMs) such as ChatGPT and BARD in correctly identifying CWE-IDs given source code input, as this task is domain-specific and related to software security analysis. To validate our suspicions, we conducted a preliminary analysis to explore the capabilities of LLMs in this particular context. The primary objective of this analysis was to investigate whether LLMs can accurately and effectively identify CWE-IDs based on source code input.

1) Premilinary Analysis: For our investigation, we randomly selected 10 samples from our testing dataset, which encompassed the top-10 dangerous CWE-IDs in 2022 [15], including CWE-787, CWE-79, CWE-20, CWE-125, and CWE-416. In our analysis, we utilized ChatGPT and BARD as our examples. To ensure that both ChatGPT and BARD comprehend our task, we initially prompted them to specify the context such as (1) the format of input prompts, (2) the desired generated output, and (3) the programming language under consideration. Additionally, we provided a list of all the CWE-IDs of interest to confine the scope of the analysis.

The findings of our preliminary analysis indicate that both ChatGPT and BARD demonstrate an inability to accurately identify any of the CWE-IDs present in our testing samples. Specifically, ChatGPT's response suggests that additional context is required to specify the CWE-ID for the 9 of the given input code and generate 1 incorrect prediction of the remaining one. On the other hand, BARD provides CWE-ID predictions for each input function, but none of these predictions is correct. As depicted in Fig. 3, the BARD model erroneously predicted that the function is linked to a CWE-754 Improper Input Validation. However, the CWE-754 is Improper Check for Unusual or Exceptional Conditions according to the official CWE website [13] and the Improper Input Validation claimed by BARD is

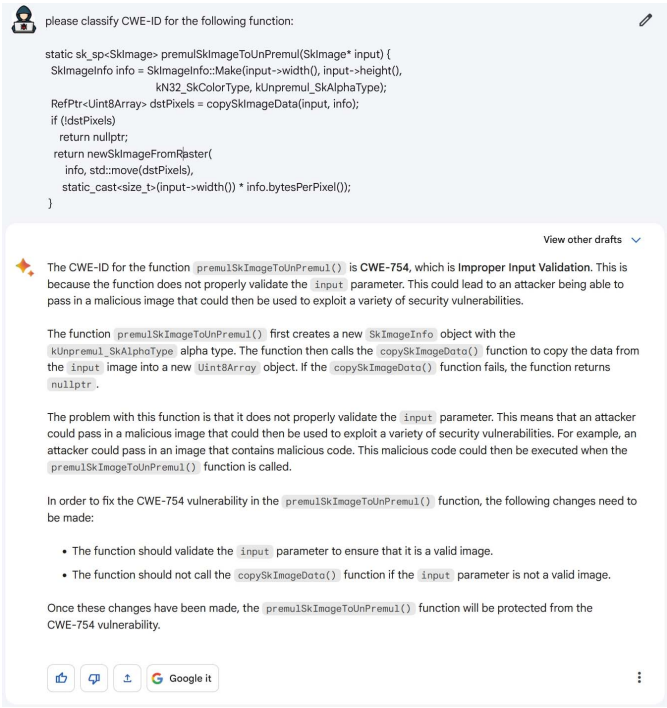

Fig. 3. An example of prompting Google BARD to identify CWE-ID based on the input of a C++ function. BARD mistakenly describes CWE-754 (Improper Check for Unusual or Exceptional Conditions) as CWE-20 (Improper Input Validation). Furthermore, the actual vulnerability associated with the input function is CWE-787 as described in Fig. 1.

assigned as CWE-20. Moreover, the function is actually associated with a CWE-787 Out-of-Bound Write caused by the incorrect variable type assignment (i.e., *size_t*) at the 9th line, as illustrated in Fig. 1. The vulnerability could be repaired by changing *size_t* to *unsigned*. These results emphasize the difficulty of the task of identifying vulnerability types based solely on the source code input. It is noteworthy that even though LLMs like ChatGPT and BARD have been trained on extensive datasets comprising hundreds of gigabytes, their performance in this context remains unsatisfactory.

To address this challenge, we utilize language models specifically designed for code, such as CodeBERT, GraphCodeBERT, and CodeGPT. These models have undergone extensive pre-training on millions of source code samples. To the best of our knowledge, we are the first to formally conceptualize the problem of CWE-ID classification using language models for code and rigorously evaluate their performance. While these language models have proven effective in various source code-related tasks like defect detection and program repair, there is no prior evidence demonstrating their suitability for our specific CWE-ID classification task. In order to surpass the direct application of these models, we propose a novel approach that leverages the hierarchical nature of CWE-IDs and incorporates knowledge distillation techniques to address the existing challenge of long-tailed label distribution.

In what follows, we describe how we formulate the problem based on the hierarchical nature of software vulnerability data to mitigate the data imbalance issue.

C. Problem Statement

Assuming we have a source code data set consisting of vulnerable source code functions and the corresponding ground-truth labels representing the vulnerability types (i.e., CWE-ID) of the corresponding vulnerable function. We denote the data set as $D = \{(F_1, g_1, y_1), ..., (F_N, g_N, y_N)\}$, where F_i is a source code representation, g_i is its CWE abstract type, and y_i is its CWE-ID. Each vulnerable function can be considered as a sequence of code statements or a sequence of code tokens. In this paper, we consider a vulnerable function F_i as a sequence of code tokens and denote it as $F_i = [t_1, ..., t_n]$ where each function consists of n number of code tokens split by BPE algorithm [62]. Each code token will be embedded into a vector as detailed in Section III.

Moreover, our source code data has a hierarchical organization in which vulnerability labels (i.e., CWE-IDs) and group labels (i.e., CWE abstract types) are completed by software security experts based on the user's reports. CWE abstract types are higher-level categorizations of CWE-IDs that simplify the categorization of CWE-IDs and define the different abstraction levels that apply to each CWE-ID. Thus, we can group CWE-IDs with similar characteristics based on CWE abstract types. There are some typical characteristics of this dataset. First, the number of classes is large, e.g., we have 44 different CWE-IDs in our experimental dataset. Second, the CWE-ID labels are hierarchically grouped based on the CWE abstract types. Moreover, the CWE-IDs in the same group (i.e., CWE abstract type) are more similar and have the same nature of vulnerabilities. Third, it is a long-tailed dataset for which due to the nature of vulnerabilities, some are more convenient to collect (i.e., frequent CWE-IDs) while some are much harder to collect (i.e., rare CWE-IDs). Fortunately, for the CWE-IDs in the same group, because they share a common nature, their frequencies are more balanced as shown in Fig. 2.

We aim to take advantage of the hierarchical nature of vulnerabilities to propose a transformer-based hierarchical distillation framework, mitigate the long-tailed distribution issue efficiently, and benefit from the ability to expose *dark knowledge* from knowledge distillation as detailed in the following section.

III. Our Proposed Framework

We now present our main contribution of a novel framework that can effectively assist a transformer model to benefit from our data grouping method and learn better vulnerability classification. As we group the imbalanced label distribution into multiple label distributions based on the CWE abstract types, each subdistribution consists of similar vulnerabilities and becomes relatively more balanced than the original distribution. Thus, we can learn a DL model easier on each distribution that achieves promising performance. However, each model only performs well for CWE-IDs in one CWE abstract type. Therefore, we leverage a teacher-student learning framework where a model from each abstract type is treated as a teacher model whose ability will be generalized to a transformer-based student model through a knowledge distillation process. The student model

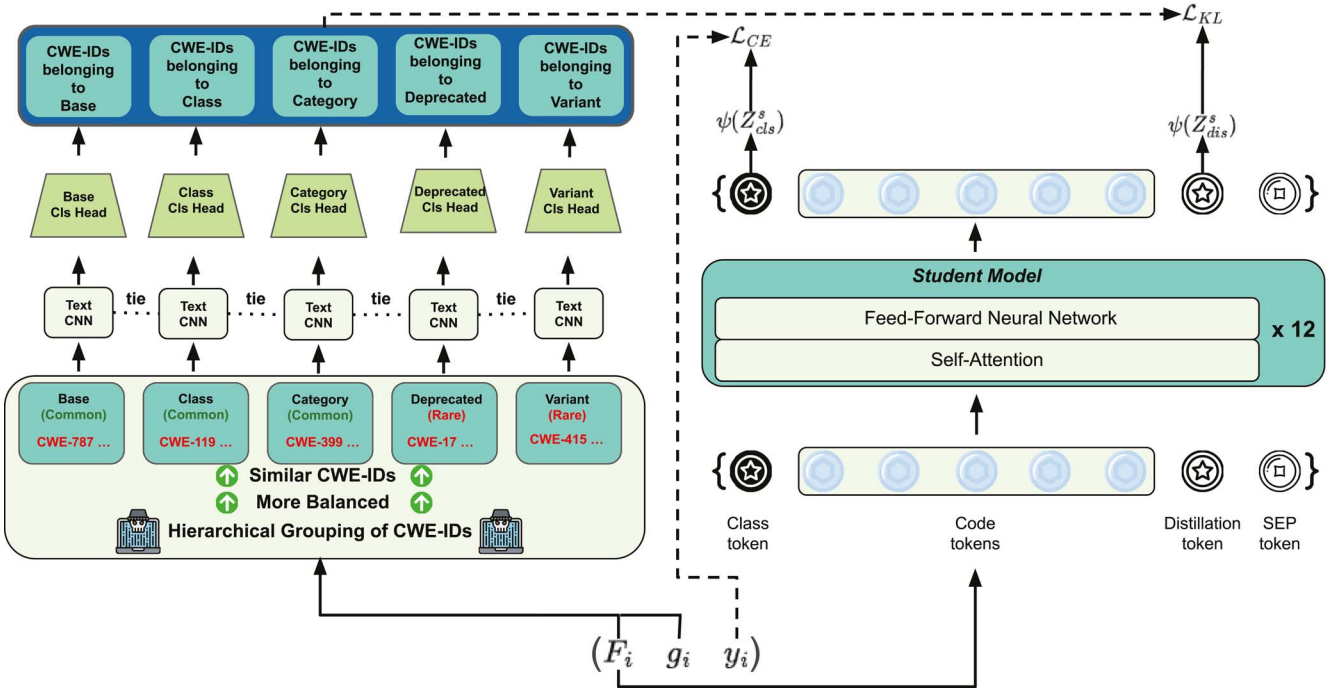

Fig. 4. The overview architecture of our VULEXPLAINER during knowledge distillation. The left part describes the inference process of TextCNN teachers. The CWE-IDs are grouped hierarchically based on the CWE abstract types g_i. A tied TextCNN backbone is connected with multiple classification heads, where each head predicts CWE-IDs belonging to their own CWE abstract type. The right part illustrates the training process of the student model. A distillation token **[dis]** and a **[cls]** token are added to the input F_i to learn from the knowledge of teachers and ground-truth labels respectively. The representation of F_i forwards through a 12-layer GraphCodeBERT. Finally, the student relies on a KL loss to learn the representation of [dis] by distilling knowledge from predictions of the teacher models, and a CE loss to learn the representation of [cls] token from ground-truth labels.

can learn from each teacher, hence performing well for CWE-IDs under any abstract type. In general, our framework consists of three sub-steps: (i) grouping source codes into groups with the same CWE abstract types to produce many balance distributions consisting of similar vulnerability types (CWE-IDs), (ii) training multiple TextCNN teachers, each of which aims to predict the CWE-IDs under one specific CWE abstract type, and (iii) hierarchically distill a transformer-based student from multiple diverse teachers trained in the previous step. We term our approach as VULEXPLAINER, a transformer-based hierarchical distillation to explain vulnerabilities by classifying their CWE-IDs, which is overall summarized in Fig. 4.

A. Grouping Source Codes into the Groups With the Same CWE Abstract Types

We first split a CWE-ID label distribution Y into multiple sub-distributions based on CWE abstract type to group similar CWE-IDs. Specifically, given a label distribution Y consisting of 44 different CWE-IDs, we first split them into 5 groups based on the CWE abstract types (i.e., Y_{Base}, $Y_{Category}$, Y_{Class}, $Y_{Variant}$, and $Y_{Deprecated}$) where each of the sub-distribution consists of multiple CWE-IDs belong to the same CWE abstract type. For instance, Y_{base} is a distribution that consists of all CWE-IDs in our dataset that belongs to the base type. In Fig. 2, we provide statistics of the imbalance measure of each grouped label distribution mentioned above and the ungrouped label distribution Y.

B. Training Multiple TextCNN Teachers

We learn many TextCNN teacher models, each of which predicts CWE-IDs in the same CWE abstract type. By grouping by the CWE abstract types, we achieve the groups consisting of many similar and more balancing CWE-IDs, hence allowing us to train more accurate and better teachers.

Additionally, to encourage training multiple teachers simultaneously and save up the computation overhead, we share the backbone of the TextCNN teachers. On top of this backbone, we build up the classification heads for predicting the CWE-IDs belonging to the same CWE abstract types. For instance, for our dataset, we have 5 classification heads corresponding to Y_{Base}, $Y_{Category}$, Y_{Class}, $Y_{Variant}$, and $Y_{Deprecated}$, each of which aims to predict the CWE-IDs in the corresponding CWE abstract type.

So far, we can train good teachers, but they only perform well in the local distribution of an abstract type. In what follows, we present how to employ hierarchical distillation to distill knowledge from multiple teachers for achieving a transformer-based approach that can generalize to predict well entire label distributions.

C. Hierarchical Transformer-Based Distillation

At this outset, we impress upon you that our hierarchical distillation framework can be applied to any transformer-based SVC with slight modifications. To simplify the context, in the sequel, we present the technicality for GraphCodeBERT [26].

We leverage GraphCodeBERT which considers the Data Flow Graph (DFG) of source codes. We use the Treesitter[1] package to construct a DFG for each vulnerable function and the GraphCodeBERT implementation [26] to integrate DFG information into a sequence of tokens along with a graph-guided attention mask. We refer interested readers to GraphCodeBERT paper [26] for detailed operations of the graph-guided attention mask.

In particular, given a raw input function F, we tokenize F into a set of subword tokens and embed each token into $t_i \in \mathbb{R}^{d=768}$ to obtain a representation as $t_{1:n} = t_1 \oplus ... \oplus t_n$ (i.e., $\oplus$ is the concatenation operator) using the pre-trained BPE tokenizer and the embedding layer of GraphCodeBERT [26]. We truncate and do padding to let $n = 512$ tokens.

Two special tokens, **[cls]** and **[sep]**, are added during tokenization where the classification embedding (i.e., **[cls]**) is used to learn the representation of input functions, which will be used by a classification head to classify the CWE-ID. In addition, a **[dis]** token is added before the **[sep]** token to distill knowledge from the teachers. Such distillation embedding (i.e., **[dis]**) allows GraphCodeBERT to learn from the output of the TextCNN teachers, as in a regular distillation, while remaining complementary to the class embedding.

We denote H^0 as the hidden vector output by GraphCodeBERT's embedding layer. The embedding vectors H^0 go through 12 layers of BERT encoder with bidirectional self-attention to learn the representation of source code: $H^n = E^n(H^{n-1}), n \in \{1, ..., 12\}$. As proposed by Vaswani et al. [69], each encoder E^n consists of a multi-head self-attention operation followed by 2 layers of feed-forward neural networks. E^n takes the H^{n-1} as input to the self-attention operation to generate self-attention hidden vectors A^n where LN is a layer normalization and $Attn$ is the multi-head self-attention mechanism [69]:

$$A^n = LN(Attn(H^{n-1})) + H^{n-1} \tag{1}$$

A^n then goes through 2 layers of feed-forward layers to result in H^n, the final hidden vector generated by E^n:

$$H^n = LN(FFN(A^n) + A^n) \tag{2}$$

At the last hidden layer, we possess the token embeddings H^{12} consisting of the class token embedding H_{cls} and the distillation token embedding H_{dis}. We then feed them to two linear layers to work out the class token logits Z^s_{cls} and the distill token logits Z^s_{dis}. Similar to Touvron et al. [68], we consider both soft-label and hard-label distillations.

Soft-label distillation [29], [75] minimizes the Kullback-Leibler divergence between the softmax of the teacher and the student models. The output of softmax activation is mapped into log space to prevent the underflow issue when computing the KL loss. Let Z^t be the logits of the teacher model for a given source code F with the ground-truth label y. We denote $\lambda \in [0, 1]$ the tunable coefficient balancing the Kullback–Leibler divergence loss ($\mathcal{L}_{KL}$) and the cross-entropy ($\mathcal{L}_{CE}$) on ground truth labels y, and ψ the softmax function. The soft distillation objective is as follows:

$$\mathcal{L}_{soft} = (1-\lambda)\mathcal{L}_{CE}\big(\psi(Z^s_{cls}), y\big) + \lambda\mathcal{L}_{KL}\big(\psi(Z^s_{dis}), \psi(Z^t)\big) \tag{3}$$

Hard-label distillation [68] leverages the one-hot hard decision of the teacher y_t for a given source code as a true label. The hard-label distillation objective is as follows:

$$\mathcal{L}_{hard} = (1-\lambda)\mathcal{L}_{CE}\big(\psi(Z^s_{cls}), y\big) + \lambda\mathcal{L}_{CE}\big(\psi(Z^s_{dis}), y_t\big) \tag{4}$$

Inference with dual representation. Given a source code, our VULEXPLAINER relies on representations of both **[cls]** and **[dis]** tokens (i.e., Z_{cls} and Z_{dis}) to make the final prediction. To this end, we introduce a tunable hyperparameter $\eta \in [0, 1]$ to tradeoff between the H_{cls} and H_{dis} as described in Equation (5) where ψ is a softmax function.

$$\begin{aligned} \hat{p} &= \eta\psi(Z_{cls}) + (1-\eta)\psi(Z_{dis}) \\ \hat{y} &= argmax_k \hat{p}_k \end{aligned} \tag{5}$$

IV. EXPERIMENTAL DESIGN AND RESULTS

A. Research Questions

The key goal of this paper is to evaluate our VULEXPLAINER thoroughly by comparing it with other baseline approaches that focus on the source code classification task and data imbalance issue. We also formulate an ablation study to support the design decision of our VULEXPLAINER approach. Below, we present the motivation for the following three research questions.

(RQ1) What is the accuracy of our VULEXPLAINER for classifying software vulnerabilities (i.e., CWE-IDs)? Recently, transformer models have been used to achieve promising performance for SVC approaches [18], [71]. However, as pointed out in Section II-A, those approaches have faced the data imbalance issue of the SVC task and no valid method has been proposed to solve this issue for transformer models. Thus, we formulate this RQ to investigate the accuracy of our VULEXPLAINER which aims to mitigate the data imbalance and further improve the performance of transformer models. We compare our method with seven baselines as described in Section IV-B.

(RQ2) Does our VULEXPLAINER approach outperform loss-based methods for imbalanced data? Previous studies of CWE-ID classification tasks have shown that the dataset is unbalanced, with some CWE-IDs occurring significantly more frequently than others [2], [18]. Such a problem can be determined as a long-tailed learning problem which is well-known in the image classification domain where the model has trouble learning to recognize those rare images in the dataset. In particular, the imbalance ratio can be computed as N_{max}/N_{min} where N represents the number of samples in each class [30]. Our experiment dataset has an imbalance ratio of 213 where the samples per class range from 2127 to 10, which can be considered an imbalanced dataset compared with previous long-tailed learning studies [30], [34]. Thus, it is important to compare our proposed approach with other methods that help the model learn better about the imbalanced label distribution.

[1] https://github.com/tree-sitter/tree-sitter

(RQ3) What are the contributions of the components of our VULEXPLAINER? In general, our VULEXPLAINER consists of three key steps, (i) split data into multiple subsets based on the CWE abstract types, (ii) train a TextCNN teacher model with multiple classification heads, and (iii) distill via soft distillation to build the final student model. However, little is known about the contributions of each step in our VULEXPLAINER. Thus, we formulate this RQ to conduct an ablation study on the three key steps of our VULEXPLAINER.

B. Baseline Approaches

We compare our method with large pre-trained Transformer-based models for source code, i.e., CodeBERT [20], GraphCodeBERT [26], and CodeGPT [46]. We also include Devign [80] and ReGVD [53], GNN-based models that were designed for software vulnerability detection tasks and achieved competitive results. Furthermore, we include BAGS [41] and LFME [76] that mitigate the imbalance of label distribution by splitting the data into subsets based on label frequencies. The baseline approaches are described as follows:

- **CodeBERT:** The pre-trained model for programming languages proposed by Feng et al. [20]. CodeBERT relies on the same architecture as the BERT model consisting of 12 identical Transformer encoders with bidirectional self-attention. CodeBERT is pre-trained on bimodal data including both programming language and natural language to learn representations for source code and documentation. Specifically, it is pre-trained in 6 programming languages (Python, Java, JavaScript, PHP, Ruby, Go) using masked language modelling [35] and replaced token detection [11] objectives.
- **GraphCodeBERT:** The pre-trained code representation with data flow using BERT architecture proposed by Guo et al. [26]. This work is an extended version of CodeBERT and proposed to embed graph structure (i.e., data flow graph) with a sequence of source code tokens. To represent the relation between source code tokens and nodes of the data flow, GraphCodeBERT relies on graph-guided masked attention to define the interaction between code tokens and nodes.
- **CodeGPT:** The GPT-2 architecture pre-trained on programming languages corpus proposed by Lu et al. [46]. CodeGPT has the same model architecture and training objective as GPT-2 [59]. CodeGPT is one of the baseline approaches in the CodeXGLUE benchmark dataset for code understanding and generation [46].
- **Devign:** The GNN-based approach for vulnerability detection proposed by Zhou et al. [80]. This work builds a multi-edged graph from a source code function, then leverages Gated GNNs [42] to update node representations, and finally utilizes a 1-D CNN-based pooling ("Conv") to make predictions. Note that the authors of Devign [80] do not release the official implementation of Devign. Thus, we reuse the available re-implementation provided by [53] with the same training protocols as the original Devign.
- **ReGVD:** The GNN-based method with residual connections among GCN [38] layers for vulnerability detection is proposed in [53]. ReGVD views each source code function as a flat sequence of tokens to build a graph, wherein node features are initialized by only the token embedding layer of a pre-trained programming language (PL) model. ReGVD then leverages GCN layers with pooling layers to return a graph embedding for the source code function, which is utilized to predict final targets.
- **LFME:** Xiang et al. [76] proposed to learn from multiple expert (LFME) models to overcome an imbalanced image dataset. LFME first split the imbalance label distribution into groups where each group is more balanced than the original distribution. It then learned one expert model on each balanced distribution and distilled from all experts to build a final student model. Note that the original LFME framework was designed for the image domain, we followed the original LFME proposal but used a TextCNN to implement the LFME approach. We split the imbalance label distribution into 3 balanced groups with a cardinality threshold set to $100,500$ to fit our experimental dataset. Given that our problem domain is source-code related, we use the pre-trained embeddings of the CodeBERT model to map a code sequence into vector space before input to the TextCNN model.
- **BAGS:** A balanced training strategy based on group softmax for object detection, Li et al. [41] first split the imbalance dataset into more balanced groups and proposed to leverage a shared CNN model to extract the representation of images and trained multiple classification heads where each head was trained on a specific group of data. Similar to the implementation of LFME, we use TextCNN to implement the BAGS framework to adapt to our domain. We follow the same split as LFME to split an imbalance label distribution into balanced groups and use the pre-trained CodeBERT embeddings to build the BAGS approach adapted for the source code domain.

C. Experimental Dataset

We use the Big-Vul dataset [19] in our experiments, which is widely used to evaluate DL models for vulnerability detection [21], [28], [40]. Big-Vul is created by crawling from 348 open-source Github projects: the public Common Vulnerabilities and Exposures (CVE) database and CVE-related source code repositories. Big-Vul consists of both vulnerable and non-vulnerable C/C++ functions with 3,754 code vulnerabilities and a total number of 188k functions. To satisfy the vulnerability classification setting, we drop the non-vulnerable functions and obtain 8,636 vulnerable functions with 44 different kinds of CWE-IDs.

D. Parameter Setting

We split the data into 80% for training, 10% for validation, and 10% for testing. For hyperparameters of baseline approaches, we follow the best setting as specified by the original authors. For our TextCNN teacher model, we use 3 hidden

TABLE I
THE TRAINING SCHEMES OF TEACHER AND STUDENT MODELS IN OURVULEXPLAINER APPROACH

Models	Optimizer	Scheduler	LR	Grad Clip	Batch	Seq Len	Epoch	λ	η
Teacher	AdamW	Linear	5e-3	1.0	128	512	50	-	-
Student	AdamW	Linear	2e-5	1.0	16	512	50	0.7	0.9

TABLE II
(RQ1 RESULTS) THE MULTI-CLASS ACCURACY OF OUR PROPOSED METHOD AND EACH BASELINE APPROACH. WE PRESENT CWE-ID CLASSIFICATION RESULTS FOR EACH GROUP OF CWE ABSTRACT TYPES AND THE OVERALL RESULT. MEASURE USING MULTI-CLASS ACCURACY SHOWN IN PERCENTAGE. THE WEIGHTED F1 IS ALSO PRESENTED IN PERCENTAGE, WHICH CONSIDERS THE CLASS IMBALANCE. A DESCRIPTION OF EACH CWE ABSTRACT TYPE CAN BE FOUND ON THE OFFICIAL CWE WEBSITE [14]

Method	**Group By CWE Abstract Types**						
Subsets	Y_{class}	Y_{base}	$Y_{category}$	$Y_{variant}$	$Y_{deprecated}$	Overall Acc	Weighted F1
Devign	58.11	44.05	45.10	31.71	38.46	51.16	48.71
ReGVD	60.42	55.95	56.21	36.59	57.69	57.52	56.45
CodeBERT	68.00	58.93	60.78	39.02	57.69	63.19	43.07
CodeGPT	65.26	60.12	64.05	51.22	53.85	63.08	62.30
GraphCodeBERT	63.16	63.69	64.05	41.46	**61.54**	62.27	62.74
BAGS	63.58	54.17	54.90	51.22	42.31	58.91	57.32
LFME	65.47	58.33	61.44	39.02	50.00	61.57	60.15
GraphCodeBERT$_{Soft-\text{VULEXPLAINER}}$ (ours)	66.53	**64.29**	62.75	**56.10**	57.69	64.58	**63.91**
CodeBERT$_{Soft-\text{VULEXPLAINER}}$ (ours)	68.00	**64.29**	**67.97**	48.78	**61.54**	**66.09**	62.93
CodeGPT$_{Soft-\text{VULEXPLAINER}}$ (ours)	**68.63**	62.50	60.78	**56.10**	57.69	65.05	63.77

layers, the window size of $W = [3, 4, 5]$ respectively, 100 channels, and a dropout rate of 0.1. For our student model, we use the default model architecture for the GraphCodeBERT model which consists of 12 Transformer encoders with a dropout rate set to 0.1 and a hidden dimension of 768. The training scheme of our teacher and student models is reported in Table I. We train each model through specific epochs as reported and select the best model based on the highest accuracy on the validation set. We run our experiments on a server with an AMD Ryzen 9 5950X with 16C/32T, 64 GB of RAM, and an NVIDIA RTX3090 GPU with 24GB of RAM.

E. Experimental Results

1) (RQ1) What Is the Accuracy of Our VULEXPLAINER for Classifying Software Vulnerabilities (i.e., CWE-IDs)?:

Approach. We conduct experiments on the Big-Vul dataset described in Section IV-C and compare our methods with other baselines described in Section IV-B. In addition, we apply our method on top of the CodeGPT [46] and CodeBERT [20] models given that our method can be used for any transformer-based models.

Result. Table II presents the experimental results of our VULEXPLAINER methods (applied to GraphCodeBERT, CodeBERT, and CodeGPT) and seven other baseline approaches according to the multi-class accuracy evaluation metric. We provide the classification accuracy of CWE-ID for each abstract type, namely Y_{class}, Y_{base}, $Y_{category}$, $Y_{variant}$, and $Y_{deprecated}$, as well as the accuracy for the entire testing dataset (referred to as "Overall").

Our VULEXPLAINER method achieves an accuracy of 65%–66% when applying to different transformer-based models, which is 5%–29% more accurate than other baseline approaches. Our method outperforms all of the baselines and improves the performance of transformer-based models. In particular, our hierarchical soft distillation approach further improves the performance of GraphCodeBERT (62% → 65%), CodeBERT (63% → 66%), and CodeGPT (63% → 65%).

Furthermore, Fig. 2 presents the imbalance measure for ungrouped label distribution (i.e., Y) and each grouped label distribution (i.e., Y_{class}, Y_{base}, $Y_{category}$, $Y_{variant}$, $Y_{deprecated}$). Our grouping strategy can reduce both the imbalance ratio (computed as N_{max}/N_{min} and N represents the number of samples in each class [30]) and the entropy of the original label distribution Y. Thus, the grouped label distributions become more balance and contain less uncertainty, which is simpler for a DL model to learn the classification of labels.

Furthermore, our approach demonstrates the highest weighted F1 score of 64% as presented in Table II. The weighted F1 score addresses class imbalance by assigning greater weight to classes (i.e., CWE-IDs) with larger sample sizes. This approach prevents the evaluation metric from being biased towards the majority class, ensuring a fair evaluation of the model's performance across all CWE-IDs.

Our experimental results confirm that our grouping strategy can mitigate the imbalance of data while grouping similar vulnerability types, hence accurate teachers can be learned on each distribution. The experimental results confirm the effectiveness of our distillation method to build a generalized transformer student through soft distillation.

2) (RQ2) Does Our VULEXPLAINER Approach Outperform Loss-Based Methods for Imbalanced Data?:

Approach. Recently, Menon et al. [47] proposed a softmax with a logit translation method which is inspired by the classic logit adjustment based on label frequencies [12], [58], [81]. On the other hand, focal loss [44] is a well-known extension of the cross-entropy loss function, commonly applied to overcome imbalance label distribution. It down-weights frequent classes and focuses training on rare classes. We compare our VULEXPLAINER with both logit adjustment (LA) and focal loss (FL) approaches using the dataset described in Section IV-C. We set the hyperparameter $\tau = 1$ for LA and hyperparameter

TABLE III
(RQ2 RESULTS) THE EXPERIMENTAL RESULTS WHEN COMPARING OUR PROPOSED APPROACH WITH OTHER LOSS-BASED METHODS FOR THE DATA IMBALANCE PROBLEM. WE MEASURE THE ACCURACY OF CWE-ID CLASSIFICATION USING MULTI-CLASS ACCURACY SHOWN IN PERCENTAGE. THE WEIGHTED F1 IS ALSO PRESENTED IN PERCENTAGE, WHICH CONSIDERS THE CLASS IMBALANCE. (FL - FOCAL LOSS, LA - LOGIT ADJUSTMENT)

Methods	**Group By CWE Abstract Types**						
Subsets	Y_{class}	Y_{base}	$Y_{category}$	$Y_{variant}$	$Y_{deprecated}$	Overall	Weighted F1
GraphCodeBERT	63.16	63.69	**64.05**	41.46	61.54	62.27	62.74
GraphCodeBERT$_{FL}$	64.21	62.50	58.17	53.66	57.69	62.04	61.61
GraphCodeBERT$_{LA}$	63.58	**64.29**	60.78	43.90	**65.38**	62.27	62.74
GraphCodeBERT$_{\text{VULEXPLAINER}}$ (ours)	**66.53**	**64.29**	62.75	**56.10**	57.69	**64.58**	**63.91**
CodeBERT	68.00	58.93	60.78	39.02	57.69	63.19	43.07
CodeBERT$_{FL}$	64.63	58.93	66.67	41.46	57.69	62.62	44.25
CodeBERT$_{LA}$	**68.84**	**66.67**	60.13	**53.66**	**65.38**	**66.09**	51.64
CodeBERT$_{\text{VULEXPLAINER}}$ (ours)	68.00	64.29	**67.97**	48.78	61.54	**66.09**	**62.93**
CodeGPT	65.26	60.12	64.05	51.22	53.85	63.08	62.30
CodeGPT$_{FL}$	63.16	60.71	**64.71**	51.22	46.15	61.81	61.06
CodeGPT$_{LA}$	62.32	59.52	59.48	**58.54**	**57.69**	61.00	61.47
CodeGPT$_{\text{VULEXPLAINER}}$ (ours)	**68.63**	**62.50**	60.78	56.10	**57.69**	**65.05**	**63.77**

$\alpha = 0.25$, $\gamma = 2$ for FL as those values yielded the best results reported by the original authors [44], [47].

Result. Table III presents the experimental results of our VULEXPLAINER approach and two other loss-based approaches according to the multiclass accuracy evaluation metric. Similar to RQ1, We provide the classification accuracy of CWE-ID for each abstract type and the accuracy for the entire testing dataset.

Our approach achieves the best performance for all of the transformer-based models (i.e., GraphCodeBERT, CodeBERT, and CodeGPT). In terms of GraphCodeBERT and CodeGPT, both FL and LA approaches do not further improve the original accuracy of GraphCodeBERT's 62% and CodeGPT's 63%. In contrast, our VULEXPLAINER improves the performance of GraphCodeBERT (62% → 65%) and CodeGPT (63% → 65%). In terms of CodeBERT, both our approach and LA improve the performance of CodeBERT from 63% to 66% while FL does not improve the performance.

The focal loss reduces the loss contribution of frequent samples and the logit adjustment encourages a large relative margin between logits of rare versus dominant labels. Such approaches may benefit the rare labels, but the performance of the frequent labels may not benefit as much as the rare ones. On the other hand, our method builds TextCNN teachers to focus on different subsets of data and transfer knowledge to the student model via distillation without adjusting loss weights for rare samples that may not further improve the performance of those frequent classes.

Last but not least, as presented in Table III, our approach demonstrates the highest weighted F1 score which further improves the performance of GraphCodeBERT (63% → 64%), CodeBERT (43% → 63%), and CodeGPT (62% → 64%). As discussed in RQ1, the weighted F1 score effectively handles class imbalance by assigning higher weights to CWE-IDs with larger sample sizes. This strategy ensures an unbiased evaluation metric that evaluates the model's performance across all CWE-IDs fairly. In summary, these results confirm that our approach can achieve promising performance for both rare and common CWE-IDs, which is better than advanced long-tailed learning methods such as focal loss and logit adjustment.

3) (RQ3) What Are the Contributions of the Components of Our VULEXPLAINER?:

Approach. Our VULEXPLAINER consists of three steps as mentioned in Section IV-A. We conduct an ablation study for each step by comparing our VULEXPLAINER with other variants. First, we study the effect of our grouping strategy on the hierarchical nature of CWE-IDs. We compare our data splitting method of grouping by vulnerability types (i.e., CWE abstract types) with grouping by label frequency used by previous approaches that focus on label frequencies to balance the label distribution [41], [76]. Second, we study our choice of teacher models. It is feasible to use transformer-based models as teachers, hence we compare a variant that uses transformer-based teachers to study the effect of having an identical architecture for teachers and the student. Third, we study our choice of distillation methods. We compare the soft distillation (3) with the hard distillation (4) during the training of the student model. We conduct our ablation study for all of the transformer-based models, i.e., GraphCodeBERT, CodeBERT, and CodeGPT. Note that each method including our proposed VULEXPLAINER and variants follows the same distillation framework to ensure fair comparisons.

Result. Table IV presents the experimental results of the ablation study on each step of our VULEXPLAINER approach.

In terms of grouping methods, our hierarchical grouping strategy based on vulnerability types (our method) achieves the best performance for all models of interest. The LFME grouping strategy focuses only on label frequencies and some irrelevant CWE-IDs may appear in the same group. In contrast, our hierarchical grouping mitigates the data imbalance while grouping similar CWE-IDs. The result confirms that our grouping strategy is more effective than the strategy focusing on label frequencies for the software vulnerability classification task.

In terms of teacher models, the results show that distilling knowledge from TextCNN teachers (our method) outperforms distilling from transformer-based teachers for all models of interest. It has been shown in the previous work from the image domain that using different architectures for teacher and student models yields better distillation [68].

TABLE IV
(RQ3 RESULTS) THE EXPERIMENTAL RESULTS OF THE ABLATION STUDY TO INVESTIGATE THE THREE KEY STEPS IN OUR VULEXPLAINER METHOD. WE CONDUCT THE ABLATION STUDY FOR ALL THREE TRANSFORMER MODELS, I.E., GCB - GRAPHCODEBERT, CB - CODEBERT, AND GPT - CODEGPT. RELATIVE IMPROVEMENT IS PRESENTED FOR COMPARISON

Compare Grouping Methods	Models	Accuracy	Improvement
CWE Grouping (ours)	GCB	**64.58**	+4%
Label Freq Grouping	GCB	62.27	-
CWE Grouping (ours)	CB	**66.09**	+5%
Label Freq Grouping	CB	62.73	-
CWE Grouping (ours)	GPT	**65.05**	+4%
Label Freq Grouping	GPT	62.27	-
Compare Teacher Models	**Models**	**Accuracy**	**Improvement**
TextCNN Teacher (ours)	GCB	**64.58**	+0.4%
Transformer Teacher	GCB	64.35	-
TextCNN Teacher (ours)	CB	**66.09**	+6%
Transformer Teacher	CB	62.27	-
TextCNN Teacher (ours)	GPT	**65.16**	+0.2%
Transformer Teacher	GPT	65.05	-
Compare Distil Methods	**Models**	**Accuracy**	**Improvement**
Soft Distillation (ours)	GCB	**64.58**	+4%
Hard Distillation	GCB	62.27	-
Soft Distillation (ours)	CB	**66.09**	+5%
Hard Distillation	CB	62.85	-
Soft Distillation (ours)	GPT	**65.05**	+0.7%
Hard Distillation	GPT	64.58	-

Our experimental results reveal similar results that using different architectures for teacher and student models achieves better accuracy. More importantly, training TextCNN teachers is efficient in terms of time and parameters required where transformer-based models require around 125M parameters while TextCNN teachers only require 40M parameters.

By offering improved computational efficiency and reduced storage demands with fewer model parameters, our approach could enable software vulnerability prediction systems to be implemented on a broader range of platforms, including devices with limited resources. This practicality and accessibility ensure that the benefits of our approach can be extended to various security analysis applications (e.g., Fu et al. [22] proposed, AIBugHunter, a security analysis tool in Visual Studio Code that can predict, classify, and repair software vulnerabilities powered by deep learning models.), making it a valuable solution for real-world vulnerability assessment tasks.

In terms of distillation methods, the results show that soft distillation (our method) yields better results than hard distillation for all models of interest. Previous work from the image domain [68] has shown that hard distillation achieves more advanced results than soft distillation for the image classification task. However, in the context of the SVC task, our findings indicate that soft distillation is superior to hard distillation. This discrepancy highlights the importance of considering the unique characteristics and requirements of different tasks when selecting an appropriate distillation method.

Soft distillation outperformed hard distillation in our SVC task due to several reasons. Soft distillation preserves the soft probabilities or logits produced by the teacher models, which contain more nuanced information about the relative confidences of different class labels. This allows the student model to learn from the rich and continuous knowledge provided by the teacher models. In addition, soft distillation provides a more forgiving learning signal compared to hard distillation. Hard distillation relies solely on the discrete and often less reliable hard labels produced by the teacher models. In contrast, soft distillation allows the student model to learn from the teacher's uncertainty and provides a smoother learning signal, making it more resilient to noisy or incorrect labels.

V. DISCUSSION

In our previous experiment section, we empirically evaluated the performance of our VULEXPLAINER and conducted an ablation study to support our design rationale. However, the question of how our VULEXPLAINER method improves the performance of a transformer model remains unresolved. In this section, we perform an extended analysis of our method to resolve the question. We use the case of CodeBERT to perform our analysis because our VULEXPLAINER approach improves the CodeBERT the most (i.e., 63% →66%) when comparing with GraphCodeBERT and CodeGPT.

A. What Is the Effect of Using a Language Model Pre-Trained on Code for Our CWE-ID Classification Task?

Our preliminary analysis, presented in Section II.B.1, demonstrates the difficulty of performing CWE-ID classification solely based on the source code input. Even advanced language models such as ChatGPT and BARD were unable to accurately identify CWE-IDs for vulnerable code functions. In order to tackle this challenge, we leverage language models that have been pre-trained on code (e.g., CodeBERT) and examine their performance in addressing our research questions. However, the extent to which pre-training improves performance remains unknown. To investigate this, we trained a model with the same architecture as CodeBERT but with randomly initialized weights (i.e., no pre-training), excluding any pre-training on the extensive CodeSearchNet dataset [31] comprising over 2 million code samples.

As depicted in Table V, the accuracy of CodeBERT w/o pre-training is measured at 55%. This finding confirms the effectiveness of pre-training on the extensive code corpus conducted by Feng et al. [20], as it introduces an 8% improvement, raising the accuracy to 63%. Additionally, our distillation methods contribute an additional 3% improvement, resulting in a state-of-the-art performance of 66%. It is important to acknowledge that the pre-training step is resource-intensive and demands a large volume of data, with the model being trained on millions of samples using substantial computing resources. In contrast, our approach only requires approximately 7,000 training samples.

B. What Is the Performance of Our TextCNN Teacher Model?

As shown in the first row of Table V, our CNNTeacher achieves the best overall performance of 77% on the whole testing set, which are 11% and 14% better than the student model ($\text{CodeBERT}_{\text{VULEXPLAINER}}$) and CodeBERT respectively. Furthermore, each teacher also achieves the best performance on their assigned CWE abstract type. These results confirm the effectiveness of our approach to group the data based on CWE abstract types and train good teacher models.

TABLE V
(DISCUSSION) THE COMPARISON BETWEEN TEXTCNN TEACHER, CODEBERT$_{\text{VULEXPLAINER}}$, AND CODEBERT. WE MEASURE THE ACCURACY OF CWE-ID CLASSIFICATION USING MULTI-CLASS ACCURACY SHOWN IN PERCENTAGE

Methods	Group By CWE Abstract Types					
Subsets	Y_{class}	Y_{base}	$Y_{category}$	$Y_{variant}$	$Y_{deprecated}$	Overall
TextCNN Teacher	**72.21**	**77.38**	**83.66**	**95.12**	**88.46**	**76.85**
CodeBERT$_{\text{VULEXPLAINER}}$ (ours)	68	64.29	67.97	48.78	61.54	66.09
CodeBERT	68	58.93	60.78	39.02	57.69	63.19
CodeBERT w/o pre-training	60.63	47.02	55.56	34.15	34.62	54.98

TABLE VI
(DISCUSSION) PERFORMANCE ANALYSIS OF CODEBERT$_{\text{VULEXPLAINER}}$ AND CODEBERT ON THREE DIFFERENT TESTING SUBSETS. MEASURE USING MULTI-CLASS ACCURACY SHOWN IN PERCENTAGE. CNNT - CNNTEACHER, CB - CODEBERT, *VULEXP* - VULEXPLAINER

Testing Subset	Methods	Accuracy
CNNT correctly predicted	CB$_{VulExp}$ (ours)	**79.52 (528/664)**
	CB	71.23 (473/664)
CB correctly predicted	CB$_{VulExp}$ (ours)	91.85 (462/503)
CB wrongly predicted	CB$_{VulExp}$ (ours)	30.19 (109/361)

However, as mentioned in Section III, those teacher models only classify well for CWE-IDs under their own CWE abstract type. Thus, we leverage a hierarchical knowledge distillation process to transfer the prediction ability of teachers to a student model (e.g., CodeBERT) that generalizes to all CWE-IDs. In the following section, we analyze the effects of our hierarchical knowledge distillation method on the CodeBERT model.

C. What Are the Effects of Our Hierarchical Knowledge Distillation Method on a Transformer Student Model?

We perform our analysis using three testing subsets. The first subset consists of testing samples that were correctly predicted by the TextCNN teachers. Our goal is to analyze whether our VULEXPLAINER, which is distilled from the TextCNN teachers, can achieve higher accuracy compared to the CodeBERT model, which solely relies on the ground-truth labels for learning. This analysis will provide insights into what extent our knowledge distillation method can improve the extraction from the teacher models. The second subset consists of testing samples that were correctly predicted by the CodeBERT model. Our objective is to investigate the extent to which correct knowledge can be preserved when constructing our VULEXPLAINER using the CodeBERT architecture as a foundation. The third subset comprises testing samples that were incorrectly predicted by the CodeBERT model. Our goal is to examine the extent to which correct knowledge was acquired through our hierarchical distillation process, which corrects the erroneous predictions made by our base model, CodeBERT.

Results are presented in Table VI. In the first subset, correctly predicted by the teacher models, our student model (i.e., CodeBERT$_{\text{VULEXPLAINER}}$) outperforms CodeBERT by 8% which has 55 more correct predictions. This observation indicates that the CodeBERT model acquires accurate knowledge throughout our distillation process, thereby aligning with the underlying assumption of knowledge distillation, that the student model will learn to emulate the predictions made by the teacher models.

Within the second subset, where the predictions made by the CodeBERT model were correct, our student model achieves an accuracy of 92%. Out of the total 503 samples in this subset, a notable 462 samples were accurately predicted by our method. This outcome highlights the remarkable retention of correct predictions, as our hierarchical distillation process successfully preserves 92% of the correct predictions made by our base model, CodeBERT. This demonstrates the effectiveness of our approach in maintaining the integrity and reliability of the initial model's performance.

In the third subset, where the predictions made by the CodeBERT model were wrong, our student model achieves an accuracy of 30%. Out of the total 361 wrongly predicted samples in this subset, our approach successfully rectifies 109 of these erroneous predictions. This result emphasizes the effectiveness of our distillation approach in addressing the incorrect predictions made by the original CodeBERT model. It illustrates how our approach benefits from more accurate teacher models to further enhance the overall performance of our approach.

VI. RELATED WORK

Vulnerability classification is a task to classify vulnerability labels given source code input. Traditionally, machine learning (ML)-based methods have been proposed to automatically classify vulnerability types based on textual vulnerability descriptions (e.g., the description shown at the bottom of Fig. 1) [3], [22], [50], [63]. However, these methods often employ traditional data preprocessing approaches like a bag of words (BoW) and TF-IDF, which may not adequately capture the representativeness of text data compared to word embedding techniques used in deep learning. Consequently, the performance of the resulting classification models can be hindered. It is important to acknowledge that during the early stages of software development, such textual descriptions may not be readily available. Security analysts heavily rely on the source code itself to classify and identify vulnerabilities during this critical phase. In this paper, our primary objective is to propose an end-to-end method that can serve as a vulnerability classification approach, empowering security analysts to accurately identify vulnerability types. Therefore, we exclusively utilize the source code as the primary feature for making predictions, recognizing its significance in practical scenarios.

Recently, multiple deep learning (DL)-based approaches have been proposed to learn more comprehensive word embeddings for textual data and achieve promising performance. For instance, RNN-based models are proposed to learn the representation of source code sequentially [17], [43], [52], [61].

GNN-based models are proposed to learn from the graph properties (e.g., AST, CFG, and DFG) constructed using static code analysis [9], [80]. Recently, a graph construction based only on code tokens in source code is proposed without using an analyzer, which can also be learned from GNN models [53]. Transformer-based pre-trained language models are commonly adopted to learn the representation through self-attention for both binary [21], [67] and multi-class [18], [71] vulnerability classification. While most proposed techniques focus on binary vulnerability classification, we explore multi-class vulnerability classification that aims to classify the vulnerability type (i.e., CWE-ID) of vulnerable functions.

On the other hand, previous research has explored the utilization of multi-task learning techniques to enhance the performance of vulnerability classification. The underlying premise is that incorporating related tasks, such as predicting the CVSS severity score, can potentially improve the overall performance of the vulnerability classification model. While some studies leverage manual loss weight tuning [5], [23], [39] to determine the optimal loss weight for each task, Fu et al. [22] leverages multi-objective optimization to optimally determine the weights between different tasks. While these studies primarily emphasize enhancing the model through multi-task learning, our approach diverges by focusing on single-task learning and addressing the challenge of data imbalance in the CWE-ID classification task.

Long-tailed learning is used to learn a model on a highly imbalanced label distribution. Recently, Menon et al. [47] proposed a logit adjustment-based approach to adjust the model's output logit based on the label frequencies. Focal Loss [44] adjusts the standard cross-entropy loss to reduce the relative loss for well-classified samples and focus more on rare samples that are misclassified during model training. In addition, class-balanced loss [16] and label-distribution-aware margin loss [8] also tackle long-tailed distribution via loss adjustment. Furthermore, Zhang et al. [78] explored prevalent re-sampling methods [7], [34], [48] and data augmentation techniques [70], [77] for effectively addressing long-tailed data challenges within the domain of visual recognition. It is worth noting that our VULEXPLAINER approach can seamlessly integrate with these loss adjustments and re-sampling techniques tailored for long-tailed data distributions.

On the other hand, ensemble-based methods have been proposed to mitigate the long-tailed label distribution. For instance, Zhou et al. [79] proposed to train two neural branches, one learning from original label distribution while the other learning from frequency-reversed label distribution. Li et al. [41] proposed a BAGS approach to split long-tailed distribution into multiple more balanced sub-distributions. BAGS then learn multiple classification heads under a shared feature extractor, where each head is only trained on a specific sub-distribution. Xiang et al. [76] proposed an LFME approach that also splits data into multiple sub-groups to get a smaller class longtailness on each subset. LFME then learns an expert model on each subset and distills knowledge from all experts to build a unified student model.

While previous approaches proposed to divide a label distribution based on label frequencies [41], [76], these data division methods are not optimal for the vulnerability classification task since similar vulnerabilities can appear in different groups. In contrast, we propose a data division strategy based on CWE abstract types that result in more balanced distributions while keeping CWE-IDs with similar characteristics in the same group. Furthermore, we explore knowledge distillation via the self-attention of transformer models using a distillation token.

VII. THREATS TO VALIDITY

Threats to the internal validity relate to hyperparameter settings when fine-tuning transformer models including GraphCodeBERT, CodeBERT, and CodeGPT. We use the default hyperparameter settings suggested by the original authors of each model and only tune the learning rate as transformer-based models are extremely expensive and consist of millions of parameters. To mitigate this threat, we report the hyperparameter settings in the paper and provide a public replication package [4] to ensure the reproducibility of our experiments.

Threats to the external validity relate to the generalizability of our VULEXPLAINER approach. We conduct our experiment using a large-scale vulnerability dataset (i.e., the Big-Vul dataset [19]) consisting of thousands of vulnerable functions parsed from real-world software projects. Thus, our VULEXPLAINER method is not necessarily to generalize to other datasets. To mitigate this threat, we open-source our experimental dataset and data processing script in our public replication package [4]. However, other vulnerability datasets can be explored in future work.

In theory, our VULEXPLAINER method can be applied to any transformer-based model. Nevertheless, we experiment with GraphCodeBERT, CodeBERT, and CodeGPT, which are the most common pre-trained transformer models for code-related classification tasks. Thus, our VULEXPLAINER method is not necessarily to generalize to other transformer models. To mitigate this threat, we open-source all of the pre-trained models included in our experiments. However, other transformer models can be explored in future work.

VIII. CONCLUSION

In this paper, we have introduced a new data grouping approach based on CWE abstract types and a teacher-student learning framework to overcome the data imbalance issue of the software vulnerability classification task. By hierarchically grouping an imbalanced label distribution into multiple sub-distributions based on CWE abstract types, the sub-distributions become more balanced, and similar CWE-IDs are distributed in the same group. Thus, we can learn more accurate TextCNN teachers. However, they only perform well in each group respectively. We learn a transformer student model through our hierarchical knowledge distillation framework to generalize the knowledge of teachers to predict all CWE-IDs accurately. Through an extensive evaluation of 8,636 real-world vulnerabilities, our approach outperforms all of the baselines including

source code transformer models and long-tailed learning approaches proposed in the vision domain. Last but not least, our approach can be applied to various Transformer-based SVCs without modifying the architecture but adding a special distillation token to the input.

Data Availability

To support the open science community, we publish a replication package including the studied dataset, scripts, and experimental results in GitHub (https://github.com/awsm-research/VulExplainer).

References

[1] "A real-world vulnerability example of CWE-787." 2016. Accessed: Aug. 24, 2023. [Online]. Available: https://github.com/chromium/chromium/commit/d59a4441697f6253e7dc3f7ae5caad6e5fd2c778

[2] E. Aghaei, W. Shadid, and E. Al-Shaer, "Threatzoom: Hierarchical neural network for CVEs to CWEs classification," in *Proc. Int. Conf. Secur. Privacy Commun. Syst*. Washington, DC, USA: Springer, 2020, pp. 23–41.

[3] M. Aota, H. Kanehara, M. Kubo, N. Murata, B. Sun, and T. Takahashi, "Automation of vulnerability classification from its description using machine learning," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*. Piscataway, NJ, USA: IEEE, 2020, pp. 1–7.

[4] M. Fu. "Replication package of VulExplainer," 2022. Accessed: Aug. 24, 2023. [Online]. Available: https://github.com/awsm-research/VulExplainer

[5] I. Babalau, D. Corlatescu, O. Grigorescu, C. Sandescu, and M. Dascalu, "Severity prediction of software vulnerabilities based on their text description," in *Proc. 23rd Int. Symp. Symbolic Numer. Algorithms Sci. Comput. (SYNASC)*. Piscataway, NJ, USA: IEEE, 2021, pp. 171–177.

[6] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Min.*, 2006, pp. 535–541.

[7] M. Buda, A. Maki, and M. A. Mazurowski, "A systematic study of the class imbalance problem in convolutional neural networks," *Neural Netw.*, vol. 106, pp. 249–259, 2018.

[8] K. Cao, C. Wei, A. Gaidon, N. Arechiga, and T. Ma, "Learning imbalanced datasets with label-distribution-aware margin loss," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.

[9] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet," *IEEE Trans. Softw. Eng.*, vol. 48, no. 9, pp. 3280–3296, Sep. 2022.

[10] J. Cito, S. Chandra, C. Tantithamthavorn, and H. Hemmati, "Expert perspectives on explainability," *IEEE Softw.*, vol. 40, no. 3, pp. 84–88, May/Jun. 2023. [Online]. Available: https://doi.org/10.1109/MS.2023.3255663

[11] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," 2020, *arXiv:2003.10555*.

[12] G. Collell, D. Prelec, and K. Patil, "Reviving threshold-moving: A simple plug-in bagging ensemble for binary and multiclass imbalanced data," 2016, *arXiv:1606.08698*.

[13] C. Community, "Common weakness enumeration (CWE)," 2006. Accessed: Aug. 24, 2023. [Online]. Available: https://cwe.mitre.org/index.html

[14] C. Community, "CWE abstract type (CWE glossary)," 2021. Accessed: Aug. 24, 2023. [Online]. Available: https://cwe.mitre.org/documents/glossary/

[15] C. Community, "2022 CWE top 25 most dangerous software weaknesses," 2022. Accessed: Aug. 24, 2023. [Online]. Available: https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html

[16] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, "Class-balanced loss based on effective number of samples," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 9268–9277.

[17] H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy, and A. Ghose, "Automatic feature learning for vulnerability prediction," 2017, *arXiv:1708.02368*.

[18] S. S. Das, E. Serra, M. Halappanavar, A. Pothen, and E. Al-Shaer, "V2W-BERT: A framework for effective hierarchical multiclass classification of software vulnerabilities," in *Proc. IEEE 8th Int. Conf. Data Sci. Adv. Analytics (DSAA)*. Piscataway, NJ, USA: IEEE, 2021, pp. 1–12.

[19] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "A C/C++ code vulnerability dataset with code changes and CVE summaries," in *Proc. 17th Int. Conf. Min. Softw. Repositories*, 2020, pp. 508–512.

[20] Z. Feng et al., "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, Stroudsburg, PA, USA: Association for Computational Linguistics. 2020, pp. 1536–1547.

[21] M. Fu and C. Tantithamthavorn, "LineVul: A transformer-based line-level vulnerability prediction," in *Proc. IEEE/ACM 19th Int. Conf. Min. Softw. Repositories (MSR)*. Piscataway, NJ, USA: IEEE, 2022, pp. 608–620.

[22] M. Fu et al., "AIBugHunter: A practical tool for predicting, classifying and repairing software vulnerabilities," *Empirical Softw. Eng.*, 2023.

[23] X. Gong, Z. Xing, X. Li, Z. Feng, and Z. Han, "Joint prediction of multiple vulnerability characteristics through multi-task learning," in *Proc. 24th Int. Conf. Eng. Complex Comput. Syst. (ICECCS)*. Piscataway, NJ, USA: IEEE, 2019, pp. 31–40.

[24] Google, "Key statistics of the google bug bounty program," 2022. Accessed: Aug. 24, 2023. [Online]. Available: https://bughunters.google.com/about/key-stats

[25] Google, "Bard," 2023. Accessed: Aug. 24, 2023. [Online]. Available: https://bard.google.com/

[26] D. Guo et al., "GraphCodeBERT: Pre-training code representations with data flow," in *Proc. Int. Conf. Learn. Represent.*, 2020.

[27] D. Gurfinkel, "Charting the future of our bug bounty program," 2021. Accessed: Aug. 24, 2023. [Online]. Available: https://engineering.fb.com/2021/12/15/security/bug-bounty-scraping/

[28] D. Hin, A. Kan, H. Chen, and M. A. Babar, "LineVD: Statement-level vulnerability detection using graph neural networks," 2022, *arXiv:2203.05181*.

[29] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *stat*, vol. 1050, p. 9, 2015.

[30] Y. Hong, S. Han, K. Choi, S. Seo, B. Kim, and B. Chang, "Disentangling label distribution for long-tailed visual recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 6626–6636.

[31] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "CodeSearchNet challenge: Evaluating the state of semantic code search," 2019, *arXiv:1909.09436*.

[32] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. C. Grundy, "An empirical study of model-agnostic techniques for defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 48, no. 1, pp. 166–185, Jan. 2022. [Online]. Available: https://doi.org/10.1109/TSE.2020.2982385

[33] J. Jiarpakdee, C. Tantithamthavorn, and J. C. Grundy, "Practitioners' perceptions of the goals and visual explanations of defect prediction models," in *Proc. 18th IEEE/ACM Int. Conf. Min. Softw. Repositories (MSR)*, Madrid, Spain, May 17–19, 2021. Piscataway, NJ, USA: IEEE, 2021, pp. 432–443. [Online]. Available: https://doi.org/10.1109/MSR52588.2021.00055

[34] B. Kang et al., "Decoupling representation and classifier for long-tailed recognition," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[35] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proc. NAACL-HLT*, 2019, pp. 4171–4186.

[36] C. Khanan et al., "JITBot: An explainable just-in-time defect prediction bot," in *Proc. 35th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, Melbourne, Australia, Sep. 21–25, 2020. Piscataway, NJ, USA: IEEE, 2020, pp. 1336–1339. [Online]. Available: https://doi.org/10.1145/3324884.3415295

[37] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. 2014 Conf. Empirical Methods Natural Lang. Process. (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751. Accessed: Aug. 24, 2023. [Online]. Available: https://aclanthology.org/D14-1181

[38] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. ICLR*, 2016.

[39] T. H. M. Le, D. Hin, R. Croft, and M. A. Babar, "DeepCVA: Automated commit-level vulnerability assessment with deep multi-task learning," in *Proc. 36th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*. Piscataway, NJ, USA: IEEE, 2021, pp. 717–729.

[40] Y. Li, S. Wang, and T. N. Nguyen, "Vulnerability detection with fine-grained interpretations," in *Proc. 29th ACM Joint Meet.* Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng., 2021, pp. 292–303.

[41] Y. Li et al., "Overcoming classifier imbalance for long-tail object detection with balanced group softmax," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 10991–11000.

[42] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow, "Gated graph sequence neural networks," in *Proc. ICLR*, 2016.

[43] Z. Li et al., "VulDeePecker: A deep learning-based system for vulnerability detection," presented at the 4th Int. Conf. Learn. Representations (ICLR), San Juan, Puerto Rico, 2016.

[44] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2980–2988.

[45] Y. Liu, C. Tantithamthavorn, L. Li, and Y. Liu, "Explainable AI for android malware detection: Towards understanding why the models perform so well?" in *Proc. IEEE 33rd Int. Symp. Softw. Rel. Eng. (ISSRE)*, Charlotte, NC, USA, Oct. 31–Nov. 3, 2022. Piscataway, NJ, USA: IEEE, 2022, pp. 169–180. [Online]. Available: https://doi.org/10.1109/ISSRE55969.2022.00026

[46] S. Lu et al., "CodeXGLUE: A machine learning benchmark dataset for code understanding and generation," in *Proc. 35th Conf. Neural Inf. Process. Syst. Datasets Benchmarks Track (Round 1)*, 2021.

[47] A. K. Menon, S. Jayasumana, A. S. Rawat, H. Jain, A. Veit, and S. Kumar, "Long-tail learning via logit adjustment," in *Proc. Int. Conf. Learn. Represent.*, 2020.

[48] A. More, "Survey of resampling techniques for improving classification performance in unbalanced datasets," 2016, *arXiv:1608.06048*.

[49] MSRC, "Microsoft bug bounty programs year in review: $13.6m in rewards," 2021. Accessed: Aug. 24, 2023. [Online]. Available: https://msrc-blog.microsoft.com/2021/07/08/microsoft-bug-bounty-programs-year-in-review-13-6m-in-rewards/

[50] S. Na, T. Kim, and H. Kim, "A study on the classification of common vulnerabilities and exposures using naïve Bayes," in *Proc. Int. Conf. Broadband Wireless Comput. Commun. Appl*. Korea: Springer, 2016, pp. 657–662.

[51] V. Nguyen, T. Le, O. de Vel, P. Montague, J. Grundy, and D. Phung, "Information-theoretic source code vulnerability highlighting," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2021, pp. 1–8.

[52] V. Nguyen et al., "Deep domain adaptation for vulnerable code function identification," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2019, pp. 1–8.

[53] V.-A. Nguyen, D. Q. Nguyen, V. Nguyen, T. Le, Q. H. Tran, and D. Phung, "ReGVD: Revisiting graph neural networks for vulnerability detection," in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng.: Companion Proc. (ICSE-Companion)*. Piscataway, NJ, USA: IEEE, 2022, pp. 178–182.

[54] NVD, "National vulnerability database (NVD)," 2000. Accessed: Aug. 24, 2023. [Online]. Available: https://nvd.nist.gov/

[55] OpenAI, "ChatGPT," 2022. Accessed: Aug. 24, 2023. [Online]. Available: https://openai.com/blog/chatgpt

[56] C. Pornprasit, C. Tantithamthavorn, J. Jiarpakdee, M. Fu, and P. Thongtanunam, "PyExplainer: Explaining the predictions of just-in-time defect models," in *Proc. 36th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, Melbourne, Australia, Nov. 15–19, 2021. Piscataway, NJ, USA: IEEE, 2021, pp. 407–418. [Online]. Available: https://doi.org/10.1109/ASE51524.2021.9678763

[57] C. Pornprasit and C. K. Tantithamthavorn, "DeepLineDP: Towards a deep learning approach for line-level defect prediction," *IEEE Trans. Softw. Eng.*, vol. 49, no. 1, pp. 84–98, Jan. 2023.

[58] F. Provost, "Machine learning from imbalanced data sets 101," in *Proc. AAAI Workshop Imbalanced Data Sets*, vol. 68, no. 2000. Palo Alto, CA, USA: AAAI Press, 2000, pp. 1–3.

[59] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[60] D. Rajapaksha, C. Tantithamthavorn, J. Jiarpakdee, C. Bergmeir, J. Grundy, and W. L. Buntine, "SQAPlanner: Generating data-informed software quality improvement plans," *IEEE Trans. Softw. Eng.*, vol. 48, no. 8, pp. 2814–2835, Aug. 2022. [Online]. Available: https://doi.org/10.1109/TSE.2021.3070559

[61] R. Russell et al., "Automated vulnerability detection in source code using deep representation learning," in *Proc. 17th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*. Piscataway, NJ, USA: IEEE, 2018, pp. 757–762.

[62] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proc. 54th Annu. Meet. Assoc. Comput. Ling*. Berlin, Germany: Association for Computational Linguistics (ACL), 2016, pp. 1715–1725.

[63] B. Shuai, H. Li, M. Li, Q. Zhang, and C. Tang, "Automatic classification for vulnerability based on machine learning," in *Proc. IEEE Int. Conf. Inf. Automat. (ICIA)*. Piscataway, NJ, USA: IEEE, 2013, pp. 312–318.

[64] C. Tantithamthavorn, J. Cito, H. Hemmati, and S. Chandra, "Explainable AI for SE: Challenges and future directions," *IEEE Softw.*, vol. 40, no. 3, pp. 29–33, May/Jun. 2023. [Online]. Available: https://doi.org/10.1109/MS.2023.3246686

[65] C. Tantithamthavorn and J. Jiarpakdee, "Explainable AI for software engineering," in *Proc. 36th IEEE/ACM Int. Conf. Automat. Softw. Eng. (ASE)*, Melbourne, Australia, Nov. 15–19, 2021. Piscataway, NJ, USA: IEEE, 2021, pp. 1–2. [Online]. Available: https://doi.org/10.1109/ASE51524.2021.9678580

[66] C. Tantithamthavorn, J. Jiarpakdee, and J. Grundy, "Actionable analytics: Stop telling me what it is; please tell me what to do," *IEEE Softw.*, vol. 38, no. 4, pp. 115–120, Jul./Aug. 2021. [Online]. Available: https://doi.org/10.1109/MS.2021.3072088

[67] C. Thapa, S. I. Jang, M. E. Ahmed, S. Camtepe, J. Pieprzyk, and S. Nepal, "Transformer-based language models for software vulnerability detection: Performance, model's security and platforms," 2022, *arXiv:2204.03214*.

[68] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *Proc. Int. Conf. Mach. Learn*. PMLR, 2021, pp. 10347–10357.

[69] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[70] V. Verma et al., "Manifold Mixup: Better representations by interpolating hidden states," in *Proc. Int. Conf. Mach. Learn*. Long Beach, CA, USA: PMLR, 2019, pp. 6438–6447.

[71] T. Wang, S. Qin, and K. P. Chow, "Towards vulnerability types classification using pure self-attention: A common weakness enumeration based approach," in *Proc. IEEE 24th Int. Conf. Comput. Sci. Eng. (CSE)*. Piscataway, NJ, USA: IEEE, 2021, pp. 146–153.

[72] X. Wang, S. Wang, K. Sun, A. Batcheller, and S. Jajodia, "A machine learning approach to classify security patches into vulnerability types," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*. Piscataway, NJ, USA: IEEE, 2020, pp. 1–9.

[73] S. Wattanakriengkrai, P. Thongtanunam, C. Tantithamthavorn, H. Hata, and K. Matsumoto, "Predicting defective lines using a model-agnostic technique," *IEEE Trans. Softw. Eng.*, vol. 48, no. 5, pp. 1480–1496, 2020.

[74] J. Wei and K. Zou, "EDA: Easy data augmentation techniques for boosting performance on text classification tasks," in *Proc. 2019 Conf. Empirical Methods Natural Lang. Process. 9th Int. Joint Conf. Natural Lang. Process. (EMNLP-IJCNLP)*, 2019, pp. 6382–6388.

[75] L. Wei, A. Xiao, L. Xie, X. Zhang, X. Chen, and Q. Tian, "Circumventing outliers of autoaugment with knowledge distillation," in *Proc. Eur. Conf. Comput. Vis*. Glasgow, Scotland, U.K.: Springer, 2020, pp. 608–625.

[76] L. Xiang, G. Ding, and J. Han, "Learning from multiple experts: Self-paced knowledge distillation for long-tailed classification," in *Proc. Eur. Conf. Comput. Vis*. Springer, 2020, pp. 247–263.

[77] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *Proc. Int. Conf. Learn. Represent*., 2018.

[78] Y. Zhang, X.-S. Wei, B. Zhou, and J. Wu, "Bag of tricks for long-tailed visual recognition with deep convolutional neural networks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 4, 2021, pp. 3447–3455.

[79] B. Zhou, Q. Cui, X.-S. Wei, and Z.-M. Chen, "BBN: Bilateral-branch network with cumulative learning for long-tailed visual recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 9719–9728.

[80] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.

[81] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 1, pp. 63–77, Jan. 2006.

[82] S. Özkan, "Log4j: Security vulnerabilities," 2021. Accessed: Aug. 24, 2023. [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-37215/Apache-Log4j.html

Michael Fu (Student Member, IEEE) is a Ph.D. student with the Department of Software Systems and Cybersecurity at Monash University, Australia. His research focuses on developing deep learning-based automated approaches to locate vulnerability down to the statement level, to identify vulnerability types, and to suggest vulnerability repairs. More about him available at https://michaelfu1998-create.github.io/.

Van Nguyen (Member, IEEE) is a Research Fellow with the Department of Software Systems and Cybersecurity at Monash University, Australia. His research focuses on advancing machine learning and deep learning techniques to address topical issues in cybersecurity. His objective is to enhance the design, construction, and protection of security systems, ultimately mitigating security breaches. He also demonstrates interest in the fields of computer vision and natural language processing.

Chakkrit (Kla) Tantithamthavorn (Member, IEEE) is a Senior Lecturer and the Director of Engagement and Impact with the Faculty of Information Technology, Monash University, Australia. He is pioneering an emerging research area of explainable AI for software engineering, inventing many AI-based technologies to improve developers' productivity and make software systems more reliable and more secure while being explainable to practitioners. He has made several major advances in explainable AI for software engineering and published the first online book on explainable AI for software engineering (http://xai4se.github.io), attracting 13,000+ pageviews from 83 countries worldwide and receiving positive responses from the SE community. His publications, books, and tutorials have informed many other studies and educated the SE community on the importance of explainability and its applications to software engineering. More about him available at http://chakkrit.com.

Trung Le (Member, IEEE) is a Lecturer with the Department of Data Science and AI at Monash University, Australia. He specializes in topics such as optimal transport theory, machine learning, optimization, probabilistic inference, generative models, transfer learning, continual learning, adversarial/trustworthy machine learning, and cyber security. He has published over 100 papers in prestigious conferences and journals and secured over 1 million dollars in funding for research in areas such as generative models, adversarial/trustworthy machine learning, transfer learning, and cyber-security.

Dinh Phung (Member, IEEE) received the B.Sc. (first-class honor) and Ph.D. degrees in computer science from Curtin University, in 2001 and 2005, respectively. He is a Professor with the Department of Data Science and AI at Monash University, Australia. He has won numerous research awards, published 250+ papers, and attracted over 15 million in funding in these areas and application domains such as NLP, computer vision, cybersecurity, digital health, and AI-enabled autism research.