HSVC: TRANSFORMER-BASED HIERARCHICAL DISTILLATION FOR SOFTWARE VULNERABILITY CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Software vulnerabilities have diverse characteristics, attacks, and impacts on software systems, stakeholders, and organizations. Such diverse characteristics of vulnerabilities (i.e., CWE-IDs) often lead to more difficulty in handling the label distributions for a Deep Learning model (e.g., addressing a highly imbalanced multi-class classification problem). However, existing vulnerability detection approaches often treat vulnerabilities equally—which does not reflect reality. In this paper, we present a new approach to solving the highly imbalanced software vulnerability classification (SVC) problem by leveraging the hierarchical structure of CWE-IDs and knowledge distillation. Specifically, we split a complex label distribution into sub-distributions based on CWE abstract types (i.e., categorizations that group similar CWE-IDs), so 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. We compare our approach with source code transformer models as well as long-tailed learning approaches proposed in the vision domain. Through an extensive evaluation using the real-world 8,636 vulnerabilities, our approach outperforms all of the baselines by 1.97%-13.89%. Our framework can be applied to any transformer-based SVC such as CodeBERT, GraphCodeBERT, and CodeGPT, with slight modifications. Training code and pre-trained models are available at `https://github.com/HSVC-TEAM/HSVC`.

1 INTRODUCTION

As the number of discovered software vulnerabilities hit an all-time high of 20k in 2021 reported by NVD (2000), large software companies are spending more and more funds mitigating the security threats by granting bug bounties (Google, 2022; MSRC, 2021; Gurfinkel, 2021). Software vulnerabilities are system weaknesses and glitches that can be further exploited by attackers to steal sensitive data or spread ransomware. Back in 2000, the National Vulnerability Database (NVD, 2000) 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, 2006) 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 (Özkan, 2021). Thus, it is important to recognize the type of vulnerability for a vulnerable program that enables the security engineer to prioritize accordingly to focus on the more severe ones.

Recently, several automated software vulnerability classifications (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 (Das et al., 2021; Wang et al.,

2021). 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 dataset, while CWE-94 is a more specific vulnerability about 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 (Lin et al., 2017; Cui et al., 2019; Cao et al., 2019; Menon et al., 2020), notably Focal Loss (Lin et al., 2017) and Logit Adjustment (Menon et al., 2020) 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 4, focal loss and logit adjustment do not improve transformer-based SVC. Additionally, some recent works have proposed to group data by label frequencies and use a balanced group softmax (Li et al., 2020) or distillation (Xiang et al., 2020) to learn a better model inspired by knowledge distillation Hinton et al. (2015) 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 2 (see the results for BAG 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.

Our proposed hierarchical distillation is based on a common characteristic of software data. Specifically, the CWE community has developed a hierarchical CWE abstract types [1] 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, 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 Figure 1. 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 (Kim, 2014) to predict the CWE-IDs in this CWE abstract type. Additionally, to save up the computation and enable 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 distil 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 (Touvron et al., 2021) for vision data. However, in our approach, we hierarchically distil from multiple TextCNN teachers based on the hierarchy of source code data.

In summary, our main contributions are: (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 distil knowledge from multiple TextCNN teachers based on the hierarchy of source code data; and (iii) a comprehensive performance evaluation of our proposed method on a large-scale standard benchmark data set including vulnerabilities from the real world.

2 Related work

Vulnerability classification is a task to classify vulnerability labels given source code input. RNN-based models are proposed to learn the representation of source code sequentially (Russell et al., 2018; Dam et al., 2017; Li et al., 2018; Nguyen et al., 2019). GNN-based models are proposed to learn from the graph properties (e.g., AST, CFG, and DFG) constructed using static code anal-

[1] https://cwe.mitre.org/documents/glossary/

ysis (Zhou et al., 2019; Chakraborty et al., 2021). 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 (Nguyen et al., 2022). Transformer-based pre-trained language models are commonly adopted to learn the representation through self-attention for both binary (Fu & Tantithamthavorn, 2022; Thapa et al., 2022) and multi-class (Das et al., 2021; Wang et al., 2021) 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.

Long-tailed learning is used to learn a model on a highly imbalanced label distribution. Recently, Menon et al. (2020) proposed a logit adjustment-based approach to adjust the model's output logit based on the label frequencies. Focal Loss (Lin et al., 2017) 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 (Cui et al., 2019) and label-distribution-aware margin loss (Cao et al., 2019) also tackle long-tailed distribution via loss adjustment.

On the other hand, ensemble-based methods have been proposed to mitigate the long-tailed label distribution. For instance, Zhou et al. (2020) proposed to train two neural branches, one learning from original label distribution while the other learning from frequency-reversed label distribution. Li et al. (2020) proposed a BAGS approach to split long-tailed distribution into multiple more balanced sub-distributions. BAGS then learns multiple classification heads under a shared feature extractor, where each head is only trained on a specific sub-distribution. Xiang et al. (2020) 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 distils knowledge from all experts to build a unified student model.

While previous approaches proposed to divide a label distribution based on label frequencies (Li et al., 2020; Xiang et al., 2020), 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 self-attention of Transformer models using a distillation token.

3 Our proposed framework

3.1 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 $D = \big\{(F_1, g_1, y_1), ..., (F_N, g_N, y_N)\big\}$, where F_i is a source code representation, g_i is its CWE abstract type, and y_i is its CWE-ID. Moreover, our source code data has a hierarchical organization in which the vulnerability label (i.e., CWE-ID) and group label (i.e., CWE abstract type) are completed by software security experts based on the user's reports.

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 as a sequence of code tokens and denote it as $F = [t_1, ..., t_n]$ where each function consist of n number of code tokens split by BPE algorithm (Sennrich et al., 2016). Each code token will be embedded into a vector as detailed in Section 3.2. 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 in 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 Table 3.

Our aim is to take advantage of these characteristics to propose a transformer-based hierarchical distillation framework to tackle the long-tailed distribution issue efficiently and benefit from the ability to expose *dark knowledge* from knowledge distillation.

3.2 PROPOSED FRAMEWORK

We now present our main contribution of a novel framework that can effectively assist a transformer model to learn better vulnerability classification. Our framework consists of three sub-steps: (i) grouping source codes into the groups with the same CWE abstract types, (ii) training multiple TextCNN teachers, each of which is to predict the CWE-IDs in the same group, and (iii) hierarchically distil a transformer-based student from multiple diverse teachers. We term this whole process as *Transformer-based Hierarchical Distillation for Software Vulnerability Classification* (HSVC), which is overall summarized in Figure 1.

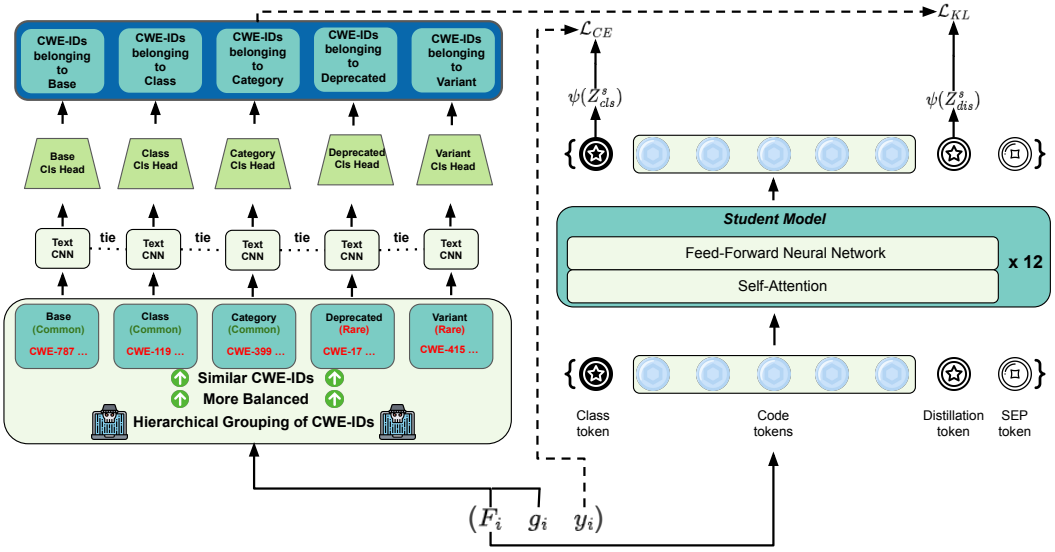

Figure 1: The overview architecture of our HSVC during knowledge distillation. The left part describes the inference process of TextCNN teachers. The CWE-IDs are grouped hierarchically based on the CWE abstract types g_i. A tied TextCNN backbone is connected with multiple classification heads, where 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.

3.2.1 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. We provide the mapping between CWE abstract types and CWE-IDs in Appendix 9. In Table 3, we provide statistics of imbalance measure of each grouped label distribution mentioned above and the ungrouped label distribution Y.

3.2.2 TRAINING MULTIPLE TEXTCNN TEACHERS

We learn multiple 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 on the local distribution of an abstract type. In what follows, we present how to employ hierarchical distillation to distil knowledge from multiple teachers for achieving a transformer-based approach that can generalize to predict well entire label distributions.

3.2.3 HIERARCHICAL TRANSFORMER-BASED DISTILLATION

At this outset, we impress upon 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 (Guo et al., 2020).

We leverage GraphCodeBERT which considers the Data Flow Graph (DFG) of source codes. We use the Treesitter [2] package to construct a DFG for each vulnerable function and the GraphCodeBERT implementation (Guo et al., 2020) to integrate DFG information into a sequence of tokens along with a graph-guided attention mask. We refer interested readers to GraphCodeBERT paper (Guo et al., 2020) 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 (Guo et al., 2020). 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 distil 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. (2017), 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 (Vaswani et al., 2017):

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

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 distil token logits Z^s_{dis}. Similar to (Touvron et al., 2021), we consider both soft-label and hard-label distillations.

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

Soft-label distillation (Hinton et al., 2015; Wei et al., 2020) 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 (Touvron et al., 2021) 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 HSVC 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.

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

4 Experiments

4.1 Baseline approaches

We compare our method with large pre-trained Transformer-based models for source code, i.e., CodeBERT (Feng et al., 2020), GraphCodeBERT Guo et al. (2020), and CodeGPT (Lu et al., 2021). We also include Devign (Zhou et al., 2019) and ReGVD (Nguyen et al., 2022), GNN-based models that were designed for software vulnerability detection tasks and achieved competitive results. Furthermore, we include BAGS (Li et al., 2020) and LFME (Xiang et al., 2020) that mitigate the imbalance of label distribution by splitting the data into subsets based on label frequencies. We refer interested readers to Appendix A.3 for more details on each baseline method.

4.2 Experimental dataset

We use Big-Vul dataset (Fan et al., 2020) in our experiments, which is widely used to evaluate DL models for vulnerability detection (Li et al., 2021; Hin et al., 2022; Fu & Tantithamthavorn, 2022). Big-Vul is created by crawling from 348 open-source Github projects which are 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.

4.3 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 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 1. 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.

Table 1: The training schemes of teacher and student models in our HSVC 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

4.4 EXPERIMENTAL RESULTS

4.4.1 MAIN RESULTS

We conduct experiments on the Big-Vul dataset to compare our methods with other baselines. In addition, we also apply our method on top of the CodeGPT (Lu et al., 2021) and CodeBERT (Feng et al., 2020). The experimental results are shown in Table 2. It can be seen that our method outperforms all of the baseline approaches. In particular, our hierarchical soft distillation approach further improves the performance of GraphCodeBERT (62.27% $\rightarrow$ 64.58%), CodeBERT (58.22% $\rightarrow$ 63.43%), and CodeGPT (63.08% $\rightarrow$ 65.05%). These results confirm the effectiveness of our proposed method to learn accurate teachers on each CWE abstract type and build a generalized Transformer student through soft distillation.

Table 2: (Main results) The multi-class accuracy on each subset of the testing data for our proposed method and each baseline approach. Measure using multi-class accuracy shown in percentage. Description of each CWE abstract type is provided in Appendix 10.

Method	Group By CWE Abstract Types					
Subsets	Y_{class}	Y_{base}	$Y_{category}$	$Y_{variant}$	$Y_{deprecated}$	Overall
Devign	58.11	44.05	45.1	31.71	38.46	51.16
ReGVD	60.42	55.95	56.21	36.59	57.69	57.52
CodeBERT	64	49.4	57.52	31.71	53.85	58.22
CodeGPT	65.26	60.12	**64.05**	51.22	53.85	63.08
GraphCodeBERT	63.16	63.69	**64.05**	41.46	**61.54**	62.27
BAGS	63.58	54.17	54.9	51.22	42.31	58.91
LFME	65.47	58.33	61.44	39.02	50	61.57
GraphCodeBERT$_{Soft-HSVC}$ (ours)	66.53	64.29	62.75	**56.1**	57.69	64.58
CodeBERT$_{Soft-HSVC}$ (ours)	65.26	**65.48**	58.82	**56.1**	57.69	63.43
CodeGPT$_{Soft-HSVC}$ (ours)	**68.63**	62.5	60.78	**56.1**	57.69	**65.05**

Additionally, we present statistics in Table 3 of the imbalance measure for each grouped label distribution (i.e., Y_{class}, Y_{base}, $Y_{category}$, $Y_{variant}$, $Y_{deprecated}$) and the ungrouped label distribution (i.e., Y). Almost all of the grouped label distributions have a lower imbalance ratio than Y, where the imbalance ratio is computed as N_{max}/N_{min} and N represents the number of samples in each class (Hong et al., 2021). Furthermore, all of the grouped label distributions have lower entropy, indicating that the grouped label distributions contain less uncertainty which is simpler for a DL model to learn the classification of labels. Such statistics confirm that our grouping strategy can mitigate the imbalance of data while grouping similar CWE-IDs.

Table 3: Statistics that measure the imbalance of grouped and ungrouped data distributions.

Measure / Distribution	Y	Y_{class}	Y_{base}	$Y_{category}$	$Y_{variant}$	$Y_{deprecated}$
Imbalance Ratio ($\frac{N_{max}}{N_{min}}$)	$\frac{2127}{10}$=213	$\frac{2127}{10}$=213	$\frac{625}{11}$=57	$\frac{736}{10}$=74	$\frac{330}{81}$=4	$\frac{177}{34}$=5
Entropy	2.78	1.66	1.98	1.25	0.5	0.85

4.4.2 ABLATION STUDY

(1) In Comparison With Recent Advanced Methods For The Data Imbalance Issue.

Previous studies on the CWE-ID classification task have revealed the data imbalance problem that vulnerable programs concentrate on specific CWE-IDs while other CWE-IDs are rare in the dataset (Das et al., 2021; Aghaei et al., 2020). 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 (Hong et al., 2021). Our experiment dataset has an imbalance ratio of 213 where the samples per class range from 2127 to 10 samples, which can be considered as an imbalance dataset when compared with previous long-tailed learning studies (Hong et al., 2021; Kang et al., 2019). Thus, it is important to compare our proposed approach with other methods that help the model learn better about the imbalanced label distribution. Recently, Menon et al. (2020) proposed a softmax with logit translation method which is inspired by the classic logit adjustment based on label frequencies (Provost, 2000; Zhou & Liu, 2005; Collell et al., 2016). On the other hand, focal loss (Lin et al., 2017) is a well-known extension of the cross-entropy loss function which is commonly applied to overcome imbalance label distribution. It down-weights frequent classes and focuses training on rare classes. We compare our HSVC with both logit adjustment (LA) and focal loss (FL) approaches. We set the hyperparameter $\tau = 1$ for LA and hyperparameter $\alpha = 0.25$, $\gamma = 2$ for FL as those values yielded the best results reported by the authors (Menon et al., 2020; Lin et al., 2017).

The experimental results are shown in Table 4. Our approach outperforms FL and LA methods by 2.54% and 2.31% respectively. When comparing with the original performance of GraphCodeBERT, both FL and LA approaches do not further improve the performance of GraphCodeBERT. In contrast, our HSVC improves the performance of GraphCodeBERT by 2.31%.

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 an approach may benefit the rare labels, but the performance on the frequent labels may not benefit as much as the rare ones. On the other hand, our method builds a TextCNN teacher with multiple classification heads to focus on different subsets of data and transfer knowledge to the student model via distillation without adjusting loss weights for rare samples.

Table 4: (Data imbalance results) The experimental results when comparing our proposed approach with other methods focusing on the data imbalance problem. Measure using multi-class accuracy shown in percentage. (FL - Focal Loss, LA - Logit Adjustment).

Methods	Group By CWE Abstract Types					
Subsets	Y_{class}	Y_{base}	$Y_{category}$	$Y_{variant}$	$Y_{deprecated}$	Overall
GraphCodeBERT	63.16	63.69	**64.05**	41.46	61.54	62.27
GraphCodeBERT$_{FL}$	64.21	62.5	58.17	53.66	57.69	62.04
GraphCodeBERT$_{LA}$	63.58	**64.29**	60.78	43.9	**65.38**	62.27
GraphCodeBERT$_{\text{HSVC}}$ (ours)	**66.53**	**64.29**	62.75	**56.1**	57.69	**64.58**

(2) Study Each Step Proposed In Our HSVC.

In general, our HSVC consists of three 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) distil via soft distillation to build the final student model. Thus, we conduct an ablation study for each step by comparing our HSVC with other variants as follows:

(i) Data Splitting.

To study the effect of our grouping strategy by the hierarchical nature of CWE-IDs, we compare our strategy with the strategy used by the LFME framework that focuses on label frequencies to balance the label distribution. We keep the same model architecture and teacher-student learning process and only change the data grouping strategy (i.e., Label Frequency Grouping + Shared TextCNN Teachers + Soft Distillation). The experimental results are shown in Table 5. The LFME grouping strategy focuses only on label frequencies and some irrelevant CWE-IDs may appear in the same group. Our hierar-

Table 5: (Ablation results) The comparison between grouping by label frequency and grouping by CWE abstract types.

Methods	Accuracy (%)
Label Frequency Grouping	62.27
CWE Grouping (ours)	**64.58**

chical grouping based on CWE abstract types 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.

(ii) Teacher Model.

We aim to study the effect of leveraging different teacher models. Thus, we compare HSVC which builds teachers sharing one TextCNN backbone with a variant that builds a separate TextCNN for each group (i.e., CWE Grouping + Non-shared TextCNN Teachers + Soft Distillation). In addition, it is feasible to use GraphCodeBERT models as teachers, hence we also compare a variant that uses GraphCodeBERT teachers (i.e., CWE Grouping + GraphCodeBERT Teachers + Soft Distillation) to study the effect of having an identical architecture for teachers and the student.

The experimental results are shown in Table 7. Our shared TextCNN teachers are more efficient in terms of computation due to fewer parameters required. Furthermore, these results confirm that distilling from shared TextCNN teachers outperforms distilling knowledge from non-shared TextCNN teachers when learning our student model. It has been shown that using different architectures for teacher and student models yields better distillation in the image domain (Touvron et al., 2021). Our experimental results also reveal 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. It is worth to note that TextCNN teachers reduce the parameters of GraphCodeBERT teachers from 125M to 40M but still achieve the same performance as shown in Table 6. Thus, we decided to leverage TextCNN teachers which achieve better performance and are more efficient to train.

Table 6: (Ablation results) The comparison between the TextCNN teachers and the Transformer teachers of our proposed method.

Teacher Models	Accuracy (%)	Params
GraphCodeBERT	75.81	125M
TextCNN	75.81	**40M**

Table 7: (Ablation result) The comparison between teachers sharing one TextCNN backbone and non-sharing teachers.

Methods	Accuracy (%)
Non-shared TextCNNs	62.73
GraphCodeBERT	64.35
Shared TextCNN (ours)	**64.58**

Table 8: (Ablation result) The comparison between the soft distillation 3 and the hard distillation 4 of our HSVC approach. Measure using multi-class accuracy shown in percentage.

Methods	Accuracy (%)
Hard Distillation	62.27
Soft Distillation (ours)	**64.58**

(iii) Distillation Method. We compare the soft distillation 3 with the hard distillation 4 (i.e., CWE Grouping + Shared TextCNN + Hard Distillation) during the training of the student model. The experimental results are shown in Table 8. Touvron et al. (2021) have shown that hard distillation achieves more advanced results than soft distillation for the image classification task. However, our experimental results show that soft distillation is better than hard distillation in the SVC task.

5 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 any transformer-based SVC with slight modifications.

REFERENCES

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

Kaidi Cao, Colin Wei, Adrien Gaidon, Nikos Arechiga, and Tengyu Ma. Learning imbalanced datasets with label-distribution-aware margin loss. *Advances in neural information processing systems*, 32, 2019.

Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. Deep learning based vulnerability detection: Are we there yet. *IEEE Transactions on Software Engineering*, 2021.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.

Guillem Collell, Drazen Prelec, and Kaustubh Patil. Reviving threshold-moving: a simple plug-in bagging ensemble for binary and multiclass imbalanced data. *arXiv preprint arXiv:1606.08698*, 2016.

Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9268–9277, 2019.

CWE. Common weakness enumeration (cwe). `https://cwe.mitre.org/index.html`, 2006.

CWE. Cwe glossary. `https://cwe.mitre.org/documents/glossary/#Variant%20Weakness`, 2021.

Hoa Khanh Dam, Truyen Tran, Trang Pham, Shien Wee Ng, John Grundy, and Aditya Ghose. Automatic feature learning for vulnerability prediction. *arXiv preprint arXiv:1708.02368*, 2017.

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

Jiahao Fan, Yi Li, Shaohua Wang, and Tien N Nguyen. Ac/c++ code vulnerability dataset with code changes and cve summaries. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pp. 508–512, 2020.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1536–1547, 2020.

Michael Fu and Chakkrit Tantithamthavorn. Linevul: A transformer-based line-level vulnerability prediction. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*. IEEE, 2022.

Google. Key statistics of the google bug bounty program. `https://bughunters.google.com/about/key-stats`, 2022.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, LIU Shujie, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. Graphcodebert: Pre-training code representations with data flow. In *International Conference on Learning Representations*, 2020.

Dan Gurfinkel. Charting the future of our bug bounty program. `https://engineering.fb.com/2021/12/15/security/bug-bounty-scraping/`, 2021.

David Hin, Andrey Kan, Huaming Chen, and M Ali Babar. Linevd: Statement-level vulnerability detection using graph neural networks. *arXiv preprint arXiv:2203.05181*, 2022.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *stat*, 1050:9, 2015.

Youngkyu Hong, Seungju Han, Kwanghee Choi, Seokjun Seo, Beomsu Kim, and Buru Chang. Disentangling label distribution for long-tailed visual recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6626–6636, 2021.

Bingyi Kang, Saining Xie, Marcus Rohrbach, Zhicheng Yan, Albert Gordo, Jiashi Feng, and Yannis Kalantidis. Decoupling representation and classifier for long-tailed recognition. In *International Conference on Learning Representations*, 2019.

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pp. 4171–4186, 2019.

Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. URL https://aclanthology.org/D14-1181.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *In ICLR*, 2016.

Yi Li, Shaohua Wang, and Tien N Nguyen. Vulnerability detection with fine-grained interpretations. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 292–303, 2021.

Yu Li, Tao Wang, Bingyi Kang, Sheng Tang, Chunfeng Wang, Jintao Li, and Jiashi Feng. Overcoming classifier imbalance for long-tail object detection with balanced group softmax. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10991–11000, 2020.

Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. Gated graph sequence neural networks. In *Proceedings of ICLR'16*, 2016.

Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. Vuldeepecker: A deep learning-based system for vulnerability detection. *arXiv e-prints*, pp. arXiv–1801, 2018.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. Codexglue: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Aditya Krishna Menon, Sadeep Jayasumana, Ankit Singh Rawat, Himanshu Jain, Andreas Veit, and Sanjiv Kumar. Long-tail learning via logit adjustment. In *International Conference on Learning Representations*, 2020.

MSRC. Microsoft bug bounty programs year in review: $13.6m in rewards. https://msrc-blog.microsoft.com/2021/07/08/microsoft-bug-bounty-programs-year-in-review-13-6m-in-rewards/, 2021.

Van Nguyen, Trung Le, Tue Le, Khanh Nguyen, Olivier DeVel, Paul Montague, Lizhen Qu, and Dinh Phung. Deep domain adaptation for vulnerable code function identification. In *International Joint Conference on Neural Networks (IJCNN)*, 2019.

Van-Anh Nguyen, Dai Quoc Nguyen, Van Nguyen, Trung Le, Quan Hung Tran, Dinh Phung, et al. Regvd: Revisiting graph neural networks for vulnerability detection. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 178–182. IEEE, 2022.

NVD. National vulnerability database (nvd). `https://nvd.nist.gov/`, 2000.

Foster Provost. Machine learning from imbalanced data sets 101. In *Proceedings of the AAAI'2000 workshop on imbalanced data sets*, volume 68, pp. 1–3. AAAI Press, 2000.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rebecca Russell, Louis Kim, Lei Hamilton, Tomo Lazovich, Jacob Harer, Onur Ozdemir, Paul Ellingwood, and Marc McConley. Automated vulnerability detection in source code using deep representation learning. In *2018 17th IEEE international conference on machine learning and applications (ICMLA)*, pp. 757–762. IEEE, 2018.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *54th Annual Meeting of the Association for Computational Linguistics*, pp. 1715–1725. Association for Computational Linguistics (ACL), 2016.

Chandra Thapa, Seung Ick Jang, Muhammad Ejaz Ahmed, Seyit Camtepe, Josef Pieprzyk, and Surya Nepal. Transformer-based language models for software vulnerability detection: Performance, model's security and platforms. *arXiv preprint arXiv:2204.03214*, 2022.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pp. 10347–10357. PMLR, 2021.

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

Tianyi Wang, Shengzhi Qin, and Kam Pui Chow. Towards vulnerability types classification using pure self-attention: A common weakness enumeration based approach. In *2021 IEEE 24th International Conference on Computational Science and Engineering (CSE)*, pp. 146–153. IEEE, 2021.

Longhui Wei, An Xiao, Lingxi Xie, Xiaopeng Zhang, Xin Chen, and Qi Tian. Circumventing outliers of autoaugment with knowledge distillation. In *European Conference on Computer Vision*, pp. 608–625. Springer, 2020.

Liuyu Xiang, Guiguang Ding, and Jungong Han. Learning from multiple experts: Self-paced knowledge distillation for long-tailed classification. In *European Conference on Computer Vision*, pp. 247–263. Springer, 2020.

Boyan Zhou, Quan Cui, Xiu-Shen Wei, and Zhao-Min Chen. Bbn: Bilateral-branch network with cumulative learning for long-tailed visual recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9719–9728, 2020.

Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in neural information processing systems*, 32, 2019.

Zhi-Hua Zhou and Xu-Ying Liu. Training cost-sensitive neural networks with methods addressing the class imbalance problem. *IEEE Transactions on knowledge and data engineering*, 18(1): 63–77, 2005.

Serkan Özkan. Log4j: Security vulnerabilities. `https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-37215/Apache-Log4j.html`, 2021.

A APPENDIX

A.1 CWE MAPPING

Table 9: The mapping between the CWE abstract types and the CWE-IDs in our benchmark data set.

CWE Abstract Type	CWE-IDs
Class	CWE-20, CWE-77, CWE-119, CWE-189, CWE-200, CWE-269, CWE-285, CWE-287, CWE-311, CWE-362, CWE-400, CWE-404, CWE-674, CWE-704, CWE-732, CWE-754, CWE-834
Base	CWE-22, CWE-59, CWE-78, CWE-79, CWE-94, CWE-125, CWE-134, CWE-190, CWE-358, CWE-369, CWE-476, CWE-617, CWE-772, CWE-787, CWE-835
Category	CWE-19, CWE-254, CWE-264, CWE-310, CWE-388, CWE-399
Variant	CWE-415, CWE-416
Deprecated	CWE-17, CWE-18, CWE-284

A.2 DESCRIPTION OF CWE ABSTRACT TYPES

Table 10: The description of each CWE abstract type (CWE, 2021) used in our benchmark data set.

CWE Abstract Type	Description
Class	A weakness that is described in a very abstract fashion, typically independent of any specific language or technology.
Base	A weakness that is described in an abstract fashion, but with sufficient details to infer specific methods for detection and prevention.
Category	A CWE entry contains a set of other entries that share a common characteristic. A category is not a weakness, but rather a structural item that helps users find weaknesses that share the stated common characteristic.
Variant	A weakness that is linked to a certain type of product, typically involving a specific language or technology.
Deprecated	A CWE entry that has been deprecated to simplify the depth and complexity of the CWE structure.

A.3 BASELINE APPROACHES

- **CodeBERT**: The pre-trained model for programming languages is proposed in (Feng et al., 2020). 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 on 6 programming languages (Python, Java, JavaScript, PHP, Ruby, Go) using masked language modelling (Kenton & Toutanova, 2019) and replaced token detection (Clark et al., 2020) objectives.
- **GraphCodeBERT**: The pre-trained code representation with data flow using BERT architecture proposed in (Guo et al., 2020). 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 is proposed in (Lu et al., 2021). CodeGPT has the same model architecture and training objective as GPT-2 (Radford et al., 2019). CodeGPT is one of the baseline approaches in the CodeXGLUE benchmark dataset for code understanding and generation (Lu et al., 2021).

- **Devign**: The GNN-based approach for vulnerability detection is proposed in (Zhou et al., 2019). This work builds a multi-edged graph from a source code function, then leverages Gated GNNs (Li et al., 2016) to update node representations, and finally utilizes a 1-D CNN-based pooling ("Conv") to make predictions. Note that the authors of Devign (Zhou et al., 2019) do not release the official implementation of Devign. Thus, we reuse the available re-implementation provided by Nguyen et al. (2022) with the same training protocols as the original Devign.
- **ReGVD**: The GNN-based method with residual connections among GCN (Kipf & Welling, 2016) layers for vulnerability detection is proposed in (Nguyen et al., 2022). 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. (2020) 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. (2020) 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.