CLeVeR: Multi-modal Contrastive Learning for Vulnerability Code Representation

Jiayuan Li[1,2], Lei Cui[3]*, Sen Zhao[1], Yun Yang[1], Lun Li[1], Hongsong Zhu[1]
[1]Institute of Information Engineering, Chinese Academy of Sciences
[2]School of Cyber Security, University of Chinese Academy of Sciences
[3]Beijing University of Posts and Telecommunications
{lijiayuan, zhaosen, yangyun, lilun, zhuhongsong}@iie.ac.cn,
cuilei@zgclab.edu.cn

Abstract

Automated vulnerability detection has become increasingly important. Many existing methods utilize deep learning models to obtain code representations for vulnerability detection. However, these approaches predominantly capture the overall semantics of the code rather than its intrinsic vulnerability-specific semantics. To address this issue, we propose CLeVeR, the first approach that leverages contrastive learning to generate precise vulnerability code representations under the supervision of vulnerability descriptions. Specifically, we introduce an Adapter, a Representation Refinement module, and a Description Simulator to mitigate the challenges of semantic misalignment and imbalance between code and descriptions, and input data inconsistency between pre-training and fine-tuning stages, respectively. For vulnerability detection and classification tasks, CLeVeR achieves F1 scores of 72.82% (real-world dataset) and 80.34%, outperforming state-of-the-art methods (SOTAs) by 11.85% and 13.61%. Additionally, CLeVeR also outperforms SOTAs in zero-shot inference, demonstrating the transferability of its generated vulnerability code representations.

1 Introduction

In recent years, with the development of deep learning (DL), DL-based code vulnerability detection has gained significant attention. Many approaches have been proposed to utilize various deep neural networks such as LSTM and BERT (Devlin et al., 2019) to learn code representations from source code (Rahman et al., 2024) and related code structures (code gadget (Li et al., 2018), SeVCs (Li et al., 2022), CPG (Chakraborty et al., 2022), PDG (Wu et al., 2022)) for vulnerability detection.

Despite the promising improvement, existing methods still suffer low precision and low recall in real-world scenarios. This is primarily because

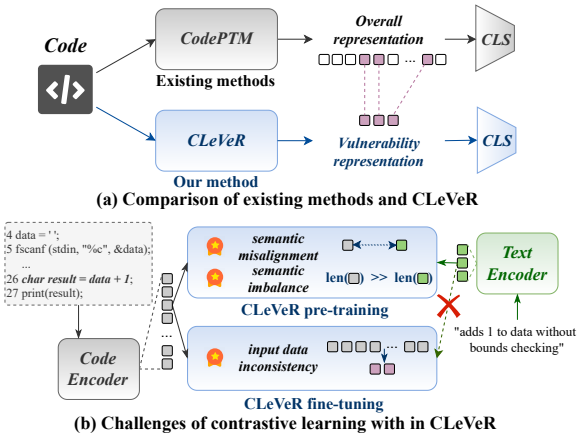

Figure 1: The Motivation of CLeVeR.

they typically learn overall code representations that capture broad functional semantics (e.g., "this code performs file parsing"), inevitably introducing noises irrelevant to vulnerability patterns, as shown in Figure 1(a). In contrast, we argue that accurately capturing vulnerability-related semantics is essential for effective detection. We observe that vulnerability descriptions contain rich vulnerability characteristics – such as "without bounds checking" – which inherently define the discriminative features we seek to capture. Thus, we propose to leverage semantic alignment techniques to utilize vulnerability description for enhancing the generation of vulnerability-specific code representations.

Leveraging generative models is one feasible approach, as it reconstructs descriptions from code. However, its reconstruction objectives tend to prioritize linguistic fluency over the detection of vulnerabilities. As a result, the model tends to focus on superficial code properties that correlate with textual patterns, rather than on the underlying vulnerability patterns. Contrastive learning, under the CLIP (Radford et al., 2021) framework, mitigates this issue by directly optimizing the semantic alignment between code and text in a shared latent space.

*Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2025, pages 7940–7951

By maximizing the mutual information of matched vulnerability pairs, the model learns more discriminative vulnerability features, avoiding the noise introduced by syntactic reconstruction.

In this paper, we propose a **C**ontrastive **Le**arning for **V**ulnerability Cod**e** **R**epresentation (CLeVeR) framework that leverages contrastive learning to acquire vulnerability code representations. Through CLIP's distinctive dual-encoder architecture and contrastive loss, CLeVeR establishes fine-grained semantic alignment between code and vulnerability descriptions. Hence, representations generated by CLeVeR are capable of capturing more precise code semantics that better characterize vulnerability patterns, which can be effectively applied in vulnerability-related downstream tasks. Although idea is simple, the direct application of CLIP faces three main challenges, as shown in Figure 1(b).

The first challenge is the semantic misalignment between different modalities, which is mainly because the variables in code and the words in descriptions are typically not in the same feature space. The second challenge arises from the semantic imbalance between different modalities. Since the full source code is much longer than the vulnerability descriptions, the semantics embedded in the code far exceed those in the descriptions, which further complicates semantic alignment. The third challenge is the input data inconsistency between pre-training and fine-tuning stages, as only source code is available during fine-tuning and testing.

CLeVeR introduces three novel components to systematically address these challenges. First, it proposes **Adapter** to project code and description representations to the same common space to improve alignment. Second, it introduces a **Representation Refinement** module that pre-refines code semantics to match the level of description semantics, enabling improved alignment. Third, it introduces the **Description Simulator**, which learns to generate refined code representations during pre-training, enabling the code representations to self-refine without descriptions. This allows CLeVeR to simulate descriptions and obtain accurate vulnerability code representations during fine-tuning. Furthermore, for evaluation, we prepared a well-annotated dataset named Vulnerability Contrastive Learning Data (VCLData)[1], which includes 280,034 vulnerable and secure functions along with corresponding descriptions, sourced from SARD (SARD).

We conduct experiments on several datasets and vulnerability-related downstream tasks. On semi-synthetic and real-world datasets for the vulnerability detection task, CLeVeR achieves F1 scores of 98.03% and 72.82%, respectively, outperforming SOTAs by 5.70% and 11.85%. For the classification task, CLeVeR attains a weighted-F1 score of 80.34%, surpassing SOTAs by 13.61%. Furthermore, CLeVeR demonstrates superior performance in zero-shot scenarios compared to previous methods. We employ visualization techniques to showcase the representations generated by CLeVeR, proving it indeed captures precise vulnerability semantics. We summarize contributions as follows:

- We propose CLeVeR, the first approach that leverages contrastive learning to generate precise vulnerability code representations under the supervision of vulnerability descriptions.
- We design the Adapter, the Representation Refinement module, and the Description Simulator to address the challenges faced in aligning semantics of code and descriptions.
- Experiments demonstrate that CLeVeR outperforms SOTA methods in vulnerability detection and classification. Our code is available at https://github.com/yoimiya-nlp/CLeVeR.

2 Related work

2.1 DL-based Vulnerability Detection

Early DL-based vulnerability detection methods (Li et al., 2018; Russell et al., 2018) utilize simple models like LSTM and CNN to capture code representations for detection. However, the detection performance of these text-based methods remains sub-optimal as they overlook the logical structure in source code. To overcome this challenge, many studies (Dam et al., 2018; Zhou et al., 2019; Wu et al., 2021, 2022; Wen et al., 2023) leverage program analysis techniques to derive both syntactic and semantic information from source code. They represent the source code with various code structures like abstract syntax trees and code property graphs (CPG), and then feed them into DL models for detection. Nevertheless, these methods focus primarily on learning overall code representations rather than specific vulnerability representations.

Recently, with the emergence of CodeBERT (Feng et al., 2020), GraphCodeBERT (Guo et al., 2021) and UniXcoder (Guo et al., 2022), methods

[1]Currently, no public dataset exists that aligns vulnerability code with natural language descriptions.

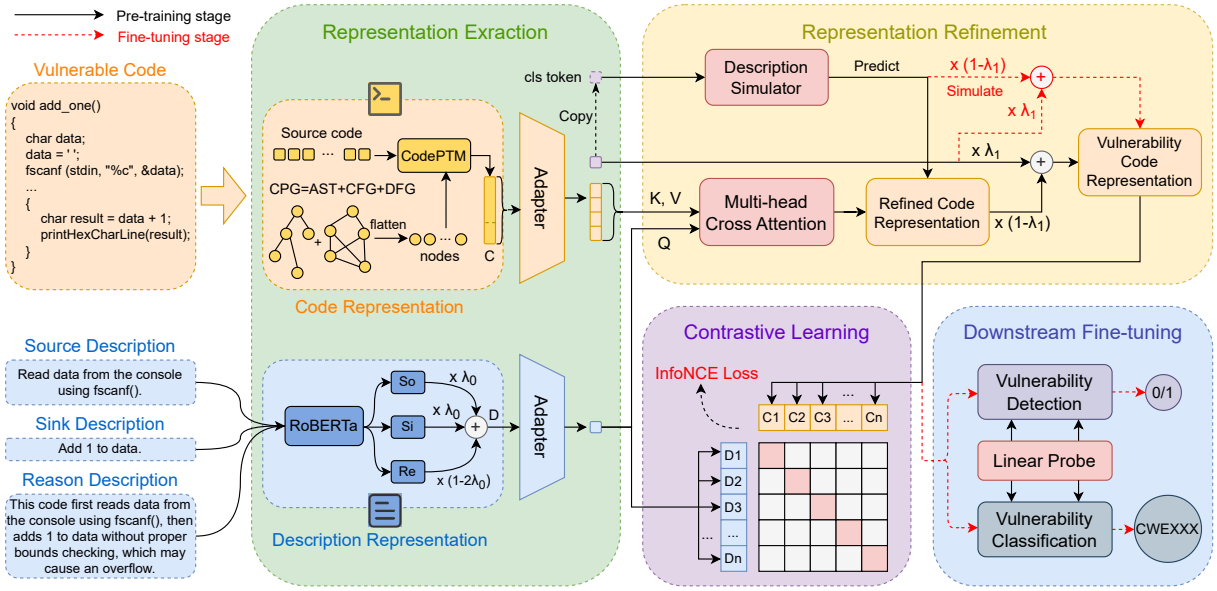

Figure 2: The overview of CLeVeR. It consists of four modules: 1) Representation Extraction module extracts both code and description representations and adapts them to the same feature space. 2) Representation Refinement module utilizes a cross-attention mechanism to generate the vulnerability code representation. 3) Contrastive Learning module compares the code and description representations to align semantics. 4) Downstream Fine-tuning module executes tasks with linear probe.

like SVulD (Ni et al., 2023) and CasualVul (Rahman et al., 2024) apply code pre-training models into vulnerability detection. However, the key challenge of extracting vulnerability-specific semantics remains unable to be addressed. In addition to detection, a few methods (Zou et al., 2019; Akshar et al., 2024; Li et al., 2025) explore classification, but they also suffer from the inability to capture vulnerability snippet representations accurately.

2.2 Contrastive Learning

Contrastive learning, as proposed in (Hadsell et al., 2006), emerges as a powerful framework in self-supervised learning, intending to learn meaningful and discriminative representations by contrasting positive and negative sample pairs. Recently, contrastive learning methods are widely applied in the domain of image and text representation learning (He et al., 2020; Chen et al., 2020; Giorgi et al., 2021; Wang et al., 2023b), and then extended to code (Cheng et al., 2022; Zamani et al., 2023). In addition, many cross-modal contrastive learning methods are proposed. For example, CLIP (Radford et al., 2021) leverages natural language supervision to learn transferable visual features, then CLAP (Wang et al., 2024) applies it to assembly code. However, vulnerability descriptions often contain less information compared to source code, posing a challenge for precise semantic alignment.

3 Methodology

The overview of CLeVeR is shown in Figure 2. In pre-training stage, we first generate the code and description representations and project them to the same feature space. Then, we refine the code representation through a cross-attention mechanism and obtain the vulnerability code representation. Consequently, we optimize the parameters of CLeVeR by comparing the vulnerability code representation with the corresponding description representation through contrastive learning, thereby obtaining a more accurate representation. In fine-tuning stage, we fine-tune the pre-trained CLeVeR model to achieve vulnerability detection and classification. Since no vulnerability descriptions are available to refine the code representation during fine-tuning and testing stages, we propose a description simulator, additionally training a feed-forward network to ensure accurate vulnerability code representation without the supervision of description.

3.1 Representation Extraction

3.1.1 Pre-trained Model

Considering raw code lacks logical information, we use CPG to augment the representation. First, we utilize the previous pre-trained models to extract representations of both code and descriptions. For the code, we choose CodeBERT (Feng et al.,

2020) to extract the sequential semantic features from source code and the syntax and logical semantic features from CPG. Specifically, we flatten CPG into a node sequence and then concatenate it with the token sequence of the source code. The relationships between nodes are represented using an adjacency matrix. The code representation is denoted as $C \in \mathbb{R}^{(l_s+l_g)\times d_c}$, where l_s and l_g represents the maximum length of the source code and the number of corresponding graph nodes, and d_c is the dimension of the code representation.

For the natural language descriptions, besides the basic cause of the vulnerability, we also consider the source and sink descriptions, which indicate where the vulnerability data is introduced and triggered, respectively. Considering these factors comprehensively can better assist in generating vulnerability code representation. We utilize a pre-trained RoBERTa (Liu et al., 2019) model to generate the representation of the natural language description, denoted as $D_{so}, D_{si}, D_{re} \in \mathbb{R}^{l_d\times d_d}$, where l_d represents the maximum length of all descriptions and d_d is the dimension of the description representation. Then, we take the value of the [CLS] token as the global contextual representation and calculate the weighted average for source, sink, and reason description representations to generate the comprehensive description representation $D = \lambda_0(D_{so}[0] + D_{si}[0]) + (1 - 2\lambda_0)D_{re}[0], D \in \mathbb{R}^{d_d}$.

3.1.2 Adapter

Pre-trained models we chose are trained within their respective modality domains, leading to substantial semantic discrepancies in the learned feature representation spaces. To ensure more consistent semantic features of both modalities, we propose a learnable Adapter following the pre-trained models. The Adapter projects the features from different modalities into a common semantic space, facilitating subsequent semantic alignment.

For the code Adapter, we integrate the sequential and logical code semantics using a similar structure with the transformer (Vaswani et al., 2017) encoder. It contains a multi-head self-attention (MHA) layer, a feed-forward network (FFN), a layer normalization (LN) layer, and a residual connection, which establish the relations between semantic information of the code tokens and the CPG nodes, generating a richer and precise code representation. The adapted code representation can be expressed as:

$$Q, K, V = W^Q C, W^K C, W^V C \quad (1)$$

$$MHA(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (2)$$

$$C = LN(FFN(MHA(Q, K, V) + C)) \quad (3)$$

where $W^Q, W^K, W^V \in \mathbb{R}^{d_c\times d_c}$ are used to calculate the query, key, and value in the attention mechanism, and d_k denotes the dimension of K.

For the description Adapter, we simply convert the description representation into the same dimensionality as the code features, denoted as d, where $d = d_c$. This conversion is implemented using a FFN and LN. The description representation can be represented as $D = LN(FFN(D))$.

3.2 Representation Refinement

Through the aforementioned module, we derived code and description representations within the same feature space. However, code representations generally encapsulate comprehensive information, whereas description representations are confined to vulnerability-specific information, posing a challenge for direct semantic alignment. Consequently, we propose a Representation Refinement module, utilizing the description representation as the query, and the code representation as the key and value. Using multi-head cross-attention (MHA), we refine the code representation, thereby generating the portion of the code representation that is relevant to the vulnerability representation. The refined code representation R can be denoted as:

$$Q, K, V = W^Q D, W^K C[1:], W^V C[1:] \quad (4)$$

$$R = MHA(Q, K, V) \quad (5)$$

where three weights $W^Q, W^K, W^V \in \mathbb{R}^{d\times d}$ are calculated in the multi-head cross-attention. To ensure that the final vulnerability code representation encompasses contextual information, we perform a weighted average of the code representations before (i.e., the value of [CLS] token in C) and after refinement, resulting in the final vulnerability code representation $V = \lambda_1 C[0] + (1 - \lambda_1)R$.

Additionally, there is a significant input data inconsistency, as no descriptions are available during both fine-tuning and testing stages on downstream tasks. To address this issue and generate accurate representations under these circumstances, we propose a Description Simulator, which learns to predict the refined representation directly from the original code representation during pre-training. During fine-tuning and testing, this simulator en-

ables the model to generate the vulnerability code representation directly from the code, just like simulating the description. Specifically, we employ a FFN and a mean squared error (MSE) loss to perform this simulator, which can be expressed as:

$$\hat{R} = FFN(C[0]) \tag{6}$$

$$\mathcal{L}_{desc} = \frac{1}{d}\sum_{k=1}^{d}(R^k - \hat{R}^k)^2 \tag{7}$$

3.3 Contrastive Learning

Inspired by the CLIP method, we apply contrastive learning to achieve semantic alignment between vulnerability code representations and their corresponding description representations. In CLeVeR, we consider the representations of a piece of vulnerability code and its corresponding description as a positive pair (R_i, D_i), while the representation of the vulnerable code and other unrelated descriptions are treated as multiple negative pairs $(R_i, D_j)_{j\neq i}$. We utilize the InfoNCE loss (van den Oord et al., 2018) to distinguish between correctly and incorrectly interpreted vulnerable code representations, which can be defined as:

$$\mathcal{L}_{info} = -\frac{1}{N}\sum_{i=1}^{N} log\frac{exp(R_i \cdot D_i)}{\sum_{j=1}^{N} exp(R_i \cdot D_j)} \tag{8}$$

where N denotes the number of samples in a single batch. By optimizing $\mathcal{L}_{info}$, we enhance the mutual information between code and vulnerability description, supremely aligning representations.

3.4 Downstream Fine-tuning

Following contrastive learning, we derive the pre-trained CLeVeR model, which can generate precise vulnerability code representation solely based on source code. To evaluate CLeVeR in vulnerability detection and classification tasks, we choose the commonly used linear probing (Fu et al., 2022), a prevalent method in representation learning, which involves freezing the pre-trained model parameters and training only a single linear projection layer. The loss functions of the pre-training stage and fine-tuning stage are denoted as:

$$\mathcal{L}_{pre-train} = \mathcal{L}_{info} + \alpha \times \mathcal{L}_{desc} \tag{9}$$

$$\mathcal{L}_{fine-tune} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}[y_{i,c}\log(\hat{y}_{i,c})] \tag{10}$$

where α is a tunable hyperparameter, N is the number of samples, $y_{i,c}$ represents the ground-truth label of sample i on category c, $\hat{y}_{i,c}$ represents the predicted results of sample i on category c, and C is the number of categories.

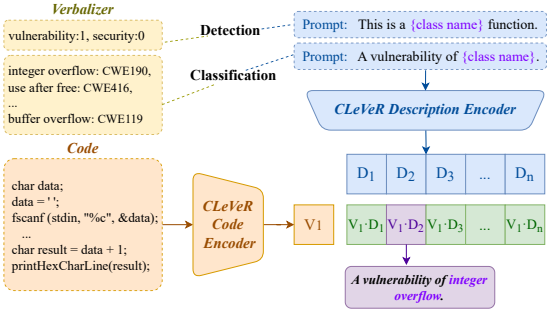

Figure 3: The zero-shot inference process of CLeVeR.

3.5 Zero-shot Inference

CLeVeR incorporates the rich semantics from natural language descriptions into vulnerability code representations, demonstrating impressive transfer capabilities. Even without fine-tuning, the vulnerability code representations generated by CLeVeR can be directly employed for zero-shot vulnerability detection and classification tasks. Figure 3 illustrates the application of zero-shot techniques in these tasks. Taking the vulnerability classification task as an example, CLeVeR first generates the vulnerability code representation V_1 for the function to be classified. Concurrently, we construct n different prompts to describe various vulnerabilities corresponding to different CWE categories, which CLeVeR processes to generate n corresponding description representations $[D_1, D_2, ..., D_n]$. We then compute the dot product between V_1 and each D_i to calculate the similarity, labeling the prompt with the highest similarity as the zero-shot classification result. Then we use the SoftMax function to acquire the probability of each category, and the kth category's probability can be expressed as:

$$P(D = D_k|V_1) = \frac{exp(V_1 \cdot D_k)}{\sum_{i=1}^{n} exp(V_1 \cdot D_i)} \tag{11}$$

4 Evaluation

4.1 Experimental Setup

4.1.1 Dataset

We construct our VCLData dataset from SARD, a widely used benchmark dataset in vulnerability detection. VCLData comprises 280,034 C/C++ functions with aligned descriptions (i.e., 77,861 vulnerable functions and 202,173 secure functions)

	VCLData-ft				SynData				RealData			
Method	A(%)	P(%)	R(%)	F1(%)	A(%)	P(%)	R(%)	F1(%)	A(%)	P(%)	R(%)	F1(%)
SySeVR (TDSC22)	86.02	91.39	69.99	79.27	87.81	91.77	79.77	85.35	65.81	55.94	42.75	48.46
Reveal (TSE22)	88.41	87.40	81.39	84.29	89.64	91.43	84.64	87.91	69.31	60.31	53.73	56.83
VulCNN (ICSE22)	87.15	92.82	71.93	81.05	86.04	83.35	85.76	84.54	67.78	59.24	45.93	51.74
AMPLE (ICSE23)	90.82	**94.50**	80.67	87.04	92.10	93.00	88.94	90.92	70.94	62.00	58.68	60.29
CodeT5 (EMNLP21)	88.25	86.04	82.68	84.32	88.71	90.37	83.52	86.81	69.78	61.40	52.84	56.80
UnixCoder (ACL22)	89.30	87.95	83.43	85.63	91.81	92.71	88.55	90.58	71.09	64.41	51.67	57.34
SVulD (FSE23)	89.36	85.16	87.35	86.25	91.29	92.27	87.78	89.97	72.05	64.61	56.77	60.43
CasualVul (ICSE24)	90.67	90.30	84.67	87.39	93.11	91.59	93.08	92.33	72.78	66.15	56.54	60.97
CLeVeR	**96.53**	92.92	**98.41**	**95.58**	**98.19**	**96.14**	**100.00**	**98.03**	**79.13**	**86.25**	**63.01**	**72.82**
CLeVeR(zero-shot)	90.63	85.55	90.81	88.11	95.49	95.51	94.42	94.96	72.04	71.81	60.87	65.89

Table 1: Comparison with SOTA vulnerability detection approaches in semi-synthetic and real-world datasets.

across 146 CWEs. We perform deduplication, manual review, and correction on VCLData. Detailed dataset processing procedures are described in Appendix A. We randomly select 80% for pre-training, denoted as VCLData-pt, while the remaining 20% is used for fine-tuning, denoted as VCLData-ft.

In addition to VCLData-ft, we evaluate CLeVeR on two traditional and widely used datasets in vulnerability detection. One is a semi-synthetic dataset proposed by (Wu et al., 2022), denoted as SynData, which contains 12,303 vulnerable functions and 21,057 secure functions. The other is the merger of two large real-world vulnerability datasets, FFMPeg+Qemu proposed by (Zhou et al., 2019) and Reveal proposed by (Chakraborty et al., 2022), denoted as RealData. RealData consists of 14,700 vulnerable functions and 35,352 secure functions. By testing on multiple diverse datasets, we can more objectively demonstrate CLeVeR's performance.

4.1.2 State-of-the-Art Approaches

We compare CLeVeR with nine SOTA methods, namely μVulDeePecker (Zou et al., 2019), SySeVR (Li et al., 2022), Reveal (Wu et al., 2022), VulCNN (Wu et al., 2022), AMPLE (Wen et al., 2023), CodeT5 (Wang et al., 2021), UnixCoder (Guo et al., 2022), SVulD (Ni et al., 2023), and CasualVul (Rahman et al., 2024). It is worth mentioning that vulnerability descriptions are only used during the pre-training stage. After pre-training, all methods perform vulnerability detection or classification solely based on the source code.

4.1.3 Configuration

We run CLeVeR on a server configured with Intel(R) Xeon(R) Gold 6230R CPU, NVIDIA A100-PCIE-80GB GPU, and Ubuntu 20.04. Unless otherwise specified, we randomly split the dataset into 70% for training, 10% for validation, and 20% for testing. We conduct the experiments five times and present the average results.

4.1.4 Metrics

We evaluate the performance in terms of five widely used metrics: Accuracy (A), Precision (P), Recall (R), F1 score (F1), and Weighted-F1.

4.2 Performance on downstream tasks

4.2.1 Performance on vulnerability detection

As reported in Table 1, CLeVeR achieves significant performance improvements across all three datasets. Compared to previous SOTAs, CLeVeR improves accuracy by 5.71%, 5.08%, and 6.35%, and enhances F1 by 8.19%, 5.70%, and 11.85%, respectively. On RealData, CLeVeR achieves a recall rate of 63.01%, surpassing the previous SOTA (i.e., AMPLE) by 4.33%. Additionally, with a precision rate of 86.25%, CLeVeR substantially reduces manual verification costs of false positives in practical vulnerability detection. On SynData, CLeVeR achieves a recall rate of 100%, demonstrating its capability to completely capture vulnerabilities in simpler scenarios. Notably, CLeVeR's performance improvement is more pronounced on real-world datasets than on semi-synthetic ones. This indicates that the vulnerability code representations generated by CLeVeR effectively capture the semantic intricacies of complex functions, enhancing its detection ability in real-world scenarios.

To exhibit the transferability of CLeVeR, we also report the zero-shot testing results in Table 1. The results demonstrate that CLeVeR outperforms all SOTAs, achieving F1 improvements of 0.72%, 2.63%, and 4.92% across three datasets, thereby validating its impressive transferability. It is worth

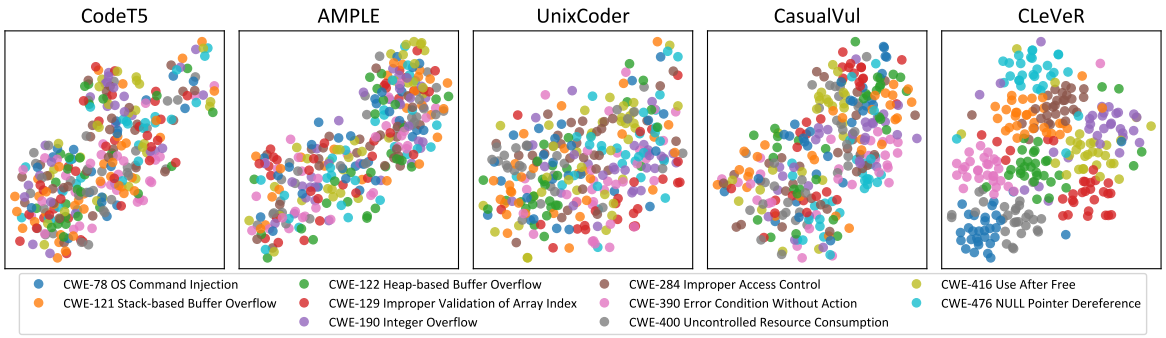

Figure 4: Intrinsic assessment using t-SNE: visualization of code representations generated by different models.

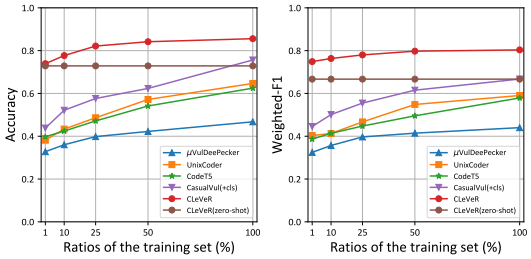

Figure 5: Comparison with SOTA vulnerability classification approaches in VCLData-ft.

noting that CLeVeR can generate precise representations while effectively avoiding overfitting issues.

4.2.2 Performance on vulnerability classification

We select the top 10 CWE categories with the highest number of correctly detected vulnerabilities by CLeVeR on the VCLData-ft dataset, totaling 9,011 vulnerabilities, for evaluation in vulnerability classification. We fine-tune the model using 1%, 10%, 25%, 50%, and 100% of the training data and also test CLeVeR in zero-shot scenarios.

As illustrated in Figure 5, with only 1% of the training data, CLeVeR achieves an accuracy of 73.90% and a Weighted-F1 of 74.87%, significantly outperforming previous methods, which demonstrates CLeVeR's excellent generalization capabilities and proficiency in few-shot learning. With 50% training data, CLeVeR achieves an accuracy of 84.16% and a Weighted-F1 of 79.75% and then increases to 85.58% and 80.34% when training on the full dataset, outperforming the previous best results by 9.95% and 13.61%, respectively. The above results indicate that CLeVeR reaches convergence more quickly and with less data compared to other models that require larger data volumes to enhance performance. Moreover, as illustrated in Figure 5, even in zero-shot scenarios, CLeVeR demonstrates comparable vulnerability classification performance to the previous best results, with an accuracy of 72.87% and a Weighted-F1 of 66.68%.

4.3 Quality of CLeVeR Representations

We conduct an intrinsic assessment to evaluate the quality of the vulnerability code representations generated by CLeVeR. We create a dataset consisting of 300 vulnerable functions by selecting 30 functions from each of the top 10 CWEs in VCLData-ft and then generate representations using CodeT5, AMPLE, UnixCoder, CasualVul, and CLeVeR. These representations are visualized in a lower-dimensional space using the t-SNE (Van der Maaten and Hinton, 2008) algorithm.

As shown in Figure 4, we observe that the code representations generated by existing approaches are chaotic and disordered from the perspective of vulnerabilities, indicating that these approaches fail to highlight the vulnerability semantics from the overall code semantics. In contrast, the representations generated by CLeVeR precisely extract and emphasize the vulnerability semantics of each function, showing that similar types of vulnerability functions have similar code representations.

4.4 Ablation study

We conduct ablation experiments on the RealData dataset to analyze the impact of CLIP and the three crucial components we proposed. First, we set a baseline by directly using CodeBERT and then evaluate the direct benefit of the generative model and contrastive learning. Second, we evaluate CLeVeR with three components individually removed to validate the contributions of each component.

As shown in Table 2, first, using CLIP leads to a greater improvement (i.e., 5.38% in F1) compared to using generative model (Sutskever et al., 2014), which shows a 2.77% improvement, validating the

Method	A(%)	P(%)	R(%)	F1(%)
Baseline	69.49	60.93	52.59	56.45
Baseline+GM	71.31	63.61	55.39	59.22
Baseline+CLIP	73.11	66.32	57.91	61.83
w/o adapter	78.44	75.91	62.50	68.56
w/o refinement	73.57	66.34	60.34	63.20
w/o simulator	76.27	71.75	60.87	65.87
CLeVeR	**79.13**	**86.25**	**63.01**	**72.82**

Table 2: Contribution of each proposed component.

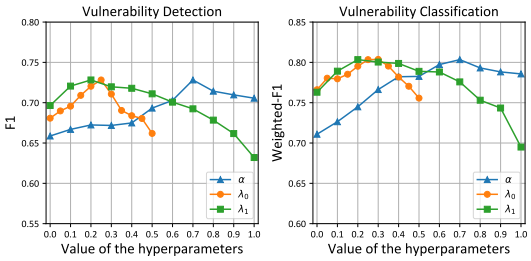

Figure 6: Comparison of varying hyperparameters.

effectiveness of CLIP. Second, all three components contribute significantly to performance improvements. Among them, the refinement module provides the largest boost, with an F1 increase of 9.62%, which suggests that the semantic imbalance between code and description representations poses a substantial challenge for semantic alignment. The simulator contributes a 6.95% F1 increase, which is less than the improvement from the refinement module. This indicates that while training the refinement module during the pre-training stage, the original code representations are also optimized. Thus, even without using the simulator to eliminate data inconsistency, the performance still improves (i.e., 2.67% in F1). Lastly, the adapter module enhances F1 by 4.26%, a relatively smaller improvement compared to other components. This is likely because both pre-training models for code and descriptions belong to the BERT family, producing inherently similar feature spaces. Its impact would be more significant when utilizing pre-training models with distinct architectures.

4.5 Hyperparameters Analysis

CLeVeR has three important hyperparameters: the weight of the description simulator α, the proportion of source and sink information in vulnerability description λ_0, and the proportion of the pre-refinement part in generating vulnerability code representations λ_1. We conduct vulnerability detection and classification tasks on the RealData and VCLData-ft datasets to evaluate CLeVeR's performance across different values of α and λ_1 in the [0,1] range, and λ_0 in the [0,0.5] range.

As illustrated in Figure 6, increasing α initially improves both vulnerability detection and classification, peaking at $\alpha = 0.7$ with an F1 score of 72.82% and a Weighted-F1 score of 80.34%. The optimal λ_0 for detection is 0.25, while for classification, it is 0.3. Analysis reveals that although setting λ_0 to 0.3 increases the Weighted-F1 score for classification by 0.03%, it decreases the F1 score for detection by 1.75%. Thus, we determined the final value of λ_0 to be 0.25 to balance both tasks. The trend for λ_1 is similar to that of α, with the best performance at $\lambda_1 = 0.2$. Notably, setting λ_1 to 0, which relies solely on refined code representation, results in a performance drop for CLeVeR. This supports the earlier assertion in Section 3.2 that incorporating contextual information is crucial for effective vulnerability code representation.

5 Discussion

The aforementioned experiments demonstrate that CLeVeR can indeed generate precise vulnerability code representations. Beyond this, we also explore other potentials of CLeVeR. We validate its computational costs, compare it with recent LLM-based methods, and examine its detection performance in real-world scenarios, thereby substantiating its potential as a practical vulnerability detection tool. Due to the page limit, we present the detailed results of these experiments in the Appendix.

6 Conclusion

In this paper, we propose CLeVeR, the first approach that utilizes contrastive learning to generate precise vulnerability code representations through vulnerability description supervision. To meet the challenges of semantic alignment, we introduce an Adapter that projects representations from disparate modalities into a unified feature space. Additionally, we design a Representation Refinement module to mitigate semantic imbalance and propose a Description Simulator to handle data inconsistency. Extensive experiments demonstrate that CLeVeR significantly outperforms all SOTA methods in several datasets and downstream tasks, with its superior performance in zero-shot scenarios further highlighting the transferability of its vulnerability code representations. This work opens up new possibilities for vulnerability detection.

Limitations

Although the CLeVeR model is capable of generating precise vulnerability code representations and performs well in vulnerability detection and classification tasks, it still has limitations. First, while the proposed VCLData dataset contains 280,034 samples across 146 CWE categories, its source is singular, as all samples are derived from SARD. Second, we have only validated the effectiveness of the CLeVeR model in vulnerability detection and classification tasks, leaving its potential for tasks such as vulnerability localization and repair unexplored. In future work, we plan to collect vulnerability data and descriptions from multiple sources, including various open-source software projects, to enrich the VCLData dataset. Additionally, we will further investigate the application of CLeVeR-generated vulnerability representations to tasks such as vulnerability localization and repair.

Ethics Statements

With the increasing prevalence of open-source software, software vulnerabilities pose significant threats to property and security, making accurate automated vulnerability detection increasingly critical. We aim to leverage deep learning techniques to obtain precise vulnerability code representations, thereby assisting security experts in detecting vulnerabilities more accurately. The pre-training dataset used in our study (VCLData-pt) was collected from SARD, an authoritative vulnerability database for vulnerability detection published by NIST (National Institute of Standards and Technology), which does not contain any sensitive or unpublished data. The testing datasets are also publicly available in vulnerability detection, and their use strictly adheres to data source usage agreements. For manual verification of VCLData, the process is conducted with four human security experts and they are all paid according to their individual working hours. After our verification, all the data we use and will open source do not contain personal or any harmful offensive information. We hope that our work contributes to maintaining a secure and reliable cyber environment.

References

Tumu Akshar, Vikram Singh, N. L. Bhanu Murthy, Aneesh Krishna, and Lov Kumar. 2024. A codebert based empirical framework for evaluating classification-enabled vulnerability prediction models. In *Proceedings of the 17th Innovations in Software Engineering Conference, ISEC 2024, Bangalore, India, February 22-24, 2024*, pages 6:1–6:11. ACM.

Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. 2022. Deep learning based vulnerability detection: Are we there yet? *IEEE Trans. Software Eng.*, 48(9):3280–3296.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. 2020. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR.

Xiao Cheng, Guanqin Zhang, Haoyu Wang, and Yulei Sui. 2022. Path-sensitive code embedding via contrastive learning for software vulnerability detection. In *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022*, pages 519–531. ACM.

Hoa Khanh Dam, Trang Pham, Shien Wee Ng, Truyen Tran, John Grundy, Aditya Ghose, Taeksu Kim, and Chul-Joo Kim. 2018. A deep tree-based model for software defect prediction. *CoRR*, abs/1802.00921.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 1536–1547. Association for Computational Linguistics.

Jinmiao Fu, Shaoyuan Xu, Huidong Liu, Yang Liu, Ning Xie, Chien-Chih Wang, Jia Liu, Yi Sun, and Bryan Wang. 2022. CMA-CLIP: cross-modality attention clip for text-image classification. In *2022 IEEE International Conference on Image Processing, ICIP 2022, Bordeaux, France, 16-19 October 2022*, pages 2846–2850. IEEE.

John M. Giorgi, Osvald Nitski, Bo Wang, and Gary D. Bader. 2021. Declutr: Deep contrastive learning for unsupervised textual representations. In *Proceedings of the 59th Annual Meeting of the Association for*

Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021, pages 879–895. Association for Computational Linguistics.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. Unixcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin B. Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. Graphcodebert: Pre-training code representations with data flow. In *9th International Conference on Learning Representations, ICLR*.

Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2006), 17-22 June 2006, New York, NY, USA*, pages 1735–1742. IEEE Computer Society.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. 2020. Momentum contrast for unsupervised visual representation learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 9726–9735. Computer Vision Foundation / IEEE.

Jiayuan Li, Lei Cui, Jie Zhang, Haiqiang Fei, Yu Chen, and Hongsong Zhu. 2025. Steering large language models for vulnerability detection. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE.

Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, and Zhaoxuan Chen. 2022. Sysevr: A framework for using deep learning to detect software vulnerabilities. *IEEE Trans. Dependable Secur. Comput.*, 19(4):2244–2258.

Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. 2018. Vuldeepecker: A deep learning-based system for vulnerability detection. In *25th Annual Network and Distributed System Security Symposium*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Guilong Lu, Xiaolin Ju, Xiang Chen, Wenlong Pei, and Zhilong Cai. 2024. GRACE: empowering llm-based software vulnerability detection with graph structure and in-context learning. *J. Syst. Softw.*, 212:112031.

Chao Ni, Xin Yin, Kaiwen Yang, Dehai Zhao, Zhenchang Xing, and Xin Xia. 2023. Distinguishing look-alike innocent and vulnerable code by subtle semantic representation learning and explanation. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, pages 1611–1622. ACM.

Yu Nong, Mohammed Aldeen, Long Cheng, Hongxin Hu, Feng Chen, and Haipeng Cai. 2024. Chain-of-thought prompting of large language models for discovering and fixing software vulnerabilities. *CoRR*, abs/2402.17230.

NVD. National Vulnerability Database (NVD). `https://nvd.nist.gov/`. Accessed: 2024-02-03.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR.

Md Mahbubur Rahman, Ira Ceka, Chengzhi Mao, Saikat Chakraborty, Baishakhi Ray, and Wei Le. 2024. Towards causal deep learning for vulnerability detection. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pages 153:1–153:11. ACM.

Rebecca L. Russell, Louis Y. Kim, Lei H. Hamilton, Tomo Lazovich, Jacob Harer, Onur Ozdemir, Paul M. Ellingwood, and Marc W. McConley. 2018. Automated vulnerability detection in source code using deep representation learning. In *17th IEEE International Conference on Machine Learning and Applications*, pages 757–762. IEEE.

SARD. SARD. `https://samate.nist.gov/SARD/`. Accessed: 2024-02-03.

Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Wei Ma, Lyuye Zhang, Miaolei Shi, and Yang Liu. 2024. Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning. *CoRR*, abs/2401.16185.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

Saad Ullah, Mingji Han, Saurabh Pujar, Hammond Pearce, Ayse K. Coskun, and Gianluca Stringhini. 2024. Llms cannot reliably identify and reason about

security vulnerabilities (yet?): A comprehensive evaluation, framework, and benchmarks. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*, pages 862–880. IEEE.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748.

Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pages 5998–6008.

Hao Wang, Zeyu Gao, Chao Zhang, Zihan Sha, Mingyang Sun, Yuchen Zhou, Wenyu Zhu, Wenju Sun, Han Qiu, and Xi Xiao. 2024. CLAP: learning transferable binary code representations with natural language supervision. *CoRR*, abs/2402.16928.

Jin Wang, Zishan Huang, Hengli Liu, Nianyi Yang, and Yinhao Xiao. 2023a. Defecthunter: A novel llm-driven boosted-conformer-based code vulnerability detection mechanism. *CoRR*, abs/2309.15324.

Yau-Shian Wang, Ta-Chung Chi, Ruohong Zhang, and Yiming Yang. 2023b. PESCO: prompt-enhanced self contrastive learning for zero-shot text classification. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 14897–14911. Association for Computational Linguistics.

Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 8696–8708. Association for Computational Linguistics.

Xin-Cheng Wen, Yupan Chen, Cuiyun Gao, Hongyu Zhang, Jie M. Zhang, and Qing Liao. 2023. Vulnerability detection with graph simplification and enhanced graph representation learning. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 2275–2286. IEEE.

Hongjun Wu, Zhuo Zhang, Shangwen Wang, Yan Lei, Bo Lin, Yihao Qin, Haoyu Zhang, and Xiaoguang Mao. 2021. Peculiar: Smart contract vulnerability detection based on crucial data flow graph and pre-training techniques. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 378–389. IEEE.

Yueming Wu, Deqing Zou, Shihan Dou, Wei Yang, Duo Xu, and Hai Jin. 2022. Vulcnn: An image-inspired scalable vulnerability detection system. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE*, pages 2365–2376. ACM.

Mahmoud Zamani, Saquib Irtiza, Shamila Wickramasuriya, Latifur Khan, and Kevin W. Hamlen. 2023. Cross domain vulnerability detection using graph contrastive learning. In *The First Tiny Papers Track at ICLR 2023, Tiny Papers @ ICLR 2023, Kigali, Rwanda, May 5, 2023*. OpenReview.net.

Yaqin Zhou, Shangqing Liu, Jing Kai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In *Annual Conference on Neural Information Processing Systems*, pages 10197–10207.

Deqing Zou, Sujuan Wang, Shouhuai Xu, Zhen Li, and Hai Jin. 2019. μ vuldeepecker: A deep learning-based system for multiclass vulnerability detection. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2224–2236.

A Detailed Dataset Processing Procedures

First, we perform hash-based deduplication on VCLData, ensuring there is no overlap between VCLData-pt and VCLData-ft. We also compare the hash values of VCLData with SynData and RealData, which shows that RealData has no overlap with VCLData-pt, while 4.12% of samples in SynData appeared in VCLData-pt. This overlap likely occurs because SynData also collected data from SARD, which explains why CLeVeR performs better on SynData than on VCLData-ft.

Second, we conduct a manual review of all 280,034 samples. We totally found 12,826 samples (4.58%) that are missing certain vulnerability descriptions (such as source or sink information), which we then supplemented. Since SARD is an official dataset, label errors were rare - we only identified 78 samples (0.03%) with incorrect labels, which were corrected after thorough discussion.

B Computation Costs of CLeVeR

The CLeVeR model contains about 0.3 billion parameters and requires just 1.5 hours of pre-training per epoch. With convergence achieved in 8-10 epochs for VCLData-pt, the total pre-training time generally remains under 15 hours. For fine-tuning, CLeVeR completes the task in less than 1 hour on the SynData dataset, which contains approximately 33,000 samples. During testing, linear probe evaluation averages only 0.112 seconds per test sample,

CVE ID	Vulnerability type	Status
CVE-2017-8309	memory leak(CWE-772)	report in NVD
CVE-2017-8380	buffer overflow(CWE-119)	report in NVD
CVE-2017-8112	infinite loop(CWE-835)	report in NVD
CVE-2017-5973	infinite loop(CWE-835)	report in NVD
CVE-2016-9922	divide by zero(CWE-369)	report in NVD
-	memory leak	no report
-	off-by-one error	no report

Table 3: Vulnerabilities detected by CLeVeR in real-world scenarios.

	SynData			RealData		
Method	A(%)	R(%)	F1(%)	A(%)	R(%)	F1(%)
GRACE	89.15	84.02	87.33	69.73	53.84	57.22
Defecthunter	88.36	82.86	86.37	67.09	50.65	53.65
VSP	89.98	86.51	89.62	73.62	54.78	60.96
LLM4Vul	88.12	83.95	87.60	66.08	55.34	55.09
CLeVeR	**98.19**	**100.00**	**98.03**	**79.13**	**63.01**	**72.82**

Table 4: Comparison with LLM-based vulnerability detection methods.

while zero-shot testing takes 0.176 seconds per sample for detection tasks and 0.426 seconds for classification tasks. Overall, these computational costs are fully manageable for practical use.

C Vulnerability Detection in Real-world Scenarios

Although we have already validated CLeVeR's vulnerability detection performance using several real-world datasets, these may not fully reflect real-world vulnerability detection scenarios. Therefore, we selected the open-source software Qemu 2.7.0 and applied the CLeVeR model pre-trained on VCLData to detect its vulnerabilities. We randomly extract 200 functions for vulnerability detection. Among the 9 vulnerabilities reported in the National Vulnerability Database (NVD) (NVD), we successfully detected 5, achieving a recall rate of 55%. In comparison, the previous best method, CasualVul, detects only 3 vulnerabilities, while UnixCoder detects just one (both methods are fine-tuned on RealData). This demonstrates CLeVeR's superior performance in real-world vulnerability detection scenarios.

Furthermore, upon analyzing the false positives detected by CLeVeR and referencing subsequent Qemu versions, we discovered 2 vulnerabilities that were modified in later versions but have not been reported in NVD. We report the 7 vulnerabilities detected by CLeVeR in Table 3. These results comprehensively illustrate CLeVeR's potential to serve as an assistant for security experts in practical vulnerability detection scenarios.

D Comparison with Recent LLM-based Vulnerability Detection Methods

With the development of large language models (LLMs) represented by GPT (OpenAI, 2023), many researchers have begun to explore the effectiveness of LLMs in vulnerability detection. However, (Ullah et al., 2024) has shown that LLMs face several challenges in vulnerability detection, including high false positive rates and unstable outputs. Nevertheless, some methods have been proposed and made certain progress. We select four recent LLM-based vulnerability detection methods (GRACE (Lu et al., 2024), Defecthunter (Wang et al., 2023a), VSP (Nong et al., 2024), LLM4Vul (Sun et al., 2024)) and evaluate their detection performance on both SynData and RealData.

As shown in Table 4, CLeVeR significantly outperforms recent LLM-based vulnerability detection methods on SynData and RealData. Specifically, CLeVeR achieves F1-score improvements of 8.39% and 12.13% respectively compared to these LLM-based methods, demonstrating its excellent vulnerability detection capability. Moreover, as mentioned in Section B, the CLeVeR model contains only about 0.3 billion parameters, requiring significantly less computational resources than LLM-based methods. In conclusion, compared to LLM-based methods, CLeVeR demonstrates superior performance in both vulnerability detection effectiveness and computational cost.