Enhanced Vulnerability Localization: Harmonizing Task-Specific Tuning and General LLM Prompting

Wentong Tian, Yuanzhang Lin, Xiang Gao*, Hailong Sun*
Beihang University, Hangzhou Innovation Institute of Beihang University
{tianwentong2000, yuanzhanglin, xiang_gao, sunhl}@buaa.edu.cn

***Abstract*—Large Language Models (LLMs) have shown significant potential for vulnerability localization in software security. However, current LLM-based approaches face a critical dilemma: direct application of general-purpose LLMs lacks crucial domain-specific expertise, while fine-tuning suffers from limited robustness when faced with unfamiliar data. These problems result in subpar performance in vulnerability localization and weak generalization capabilities. To address these limitations, we introduce ENVUL, a novel domain adaptation framework for vulnerability localization. ENVUL improves vulnerability localization by synergizing enhanced task-specific tuning with prompt engineering of general-purpose LLMs. ENVUL incorporates three key innovations for addressing two problems: (1) *how to optimize fine-tuning for localization task*, and (2) *when to wisely choose tuning and prompting*. To solve the first problem, we introduce: (a). a context *Consolidator* that captures rich statement-level code semantic, improving the model's understanding of code context; (b). a semantic *Indicator* employing attention rectification to highlight patterns indicative of vulnerabilities, focusing the model on critical security signals. To solve the second problem, we introduce a dynamic routing mechanism based on joint-representation similarity analysis that strategically delegates tasks between the fine-tuned model and the general LLM. It ensures ENVUL's robust performance across diverse real-world vulnerability types. Real-world evaluations demonstrate ENVUL's robust expertise in outperforming state-of-the-art vulnerability localization baselines, achieving absolute improvements of 22.7%-30.3% in top-1 accuracy. Notably, ENVUL exhibits exceptional generalization, achieving 43.6%-50% higher accuracy on unfamiliar vulnerability types.**

***Index Terms*—Software Vulnerability, Vulnerability Localization, Large Language Model, Domain Analysis.**

I. INTRODUCTION

Accurate vulnerability localization is essential for effective remediation, driving the need for automated techniques. The increasing prevalence of disclosed vulnerabilities, highlighted by academic [1]–[12] and industrial efforts [13], [14], underscores the urgency of advanced solutions. Large Language Models (LLMs), such as Claude-3.5 [15], GPT-4o [16], and Codellama [17], have emerged as a promising avenue. LLM-based vulnerability localization is mainly divided into two categories: fine-tuning and prompting. Fine-tuning technique [7], [18], [19] utilizes vulnerability data to adapt models for vulnerability localization. Prompt engineering [20], [21] guides LLM via task-specific prompts to identify the vulnerability location in given inputs. However, these LLM-based approaches still suffer from several limitations.

- **Prompt Engineering with Limited Domain Expertise** General-purpose LLMs, such as GPT-4o [16] and Claude-3.5 sonnet [15], demonstrate significant versatility in handling a wide range of tasks, including vulnerability localization. However, these models lack the domain-specific expertise required for high accuracy in specialized tasks like vulnerability detection. Consequently, their performance in vulnerability localization often falls short of that achieved by fine-tuned models.
- **Fine-tuning with Narrow Knowledge and Insufficient Semantic** To address the domain expertise gap in general-purpose LLMs, existing methods like [19] fine-tune an adapter for localization task. While this boosts accuracy for domain-specific vulnerability, it inadvertently narrows the model's knowledge scope on the training dataset. Much work [22]–[24] has shown that fine-tuned model is prone to provide overly-confident and wrong responses for inputs that exceed models' knowledge scope [25]. Therefore, due to discrepancies between the training data and real-world scenarios, it is infeasible for fine-tuned models to encompass all vulnerability types. This often leads to the mislocalization of unfamiliar vulnerabilities. Moreover, the scarcity of comprehensive vulnerability data further constrains the breadth of knowledge acquired by fine-tuned models. From program analysis perspective, vulnerability localization requires understanding statement-level program semantics, yet current approaches lack the ability to comprehend global semantics at the statement level, hindering accuracy. Furthermore, existing techniques struggle to discriminate critical intra-statement code elements, such as the specific integer types ('int', 'short') that are central to vulnerabilities like CWE-190: Integer Overflow or Wraparound.

Although tune-based and prompt-based techniques have their respective limitations, their characteristics are complementary. Specifically, fine-tuning techniques can provide the domain expertise that general LLMs lack, while prompting methods can address the limited breadth of knowledge in fine-tuned models. However, realizing this complementary relationship presents a critical dilemma in practical applications: *how to make an intelligent choice between fine-tuning and prompting?*

*Corresponding authors.

To address the aforementioned problems, this paper introduces a novel approach, named ENVUL, to localize vulnerabilities. The objectives of ENVUL are twofold: 1) enhancing domain-specific fine-tuning performance in vulnerability localization tasks. 2) improving the robustness to handle unfamiliar vulnerabilities by bridging the gap between tuning and prompting.

First, to improve task-specific fine-tuning performance, ENVUL integrates a *consolidator* and a *indicator* to comprehensively represent statement-level semantics. The indicator utilizes an adaptive domain mask for pruning spurious parametric feature, and an attention rectifier mechanism to precisely pinpoint and emphasize the underlying vulnerability patterns within each statement. The consolidator collects comprehensive token-level representations for each statement, explicitly capturing both intra-statement and inter-statement dependencies. The two modules enable the fine-tuned model to understand the semantic nuances within and between statements more deeply. However, due to limited training data, the enhanced fine-tuned model can still produce false positives when faced with unfamiliar vulnerabilities. Therefore, to improve the tool's robustness against seen and unseen vulnerabilities, ENVUL incorporates a novel dynamic routing mechanism that coordinates task handling between the fine-tuned model and the general LLM strategically. It categorizes vulnerabilities as domain-specific or out-of-domain, and directs domain-specific data to the task-enhanced tuned model to maximize the precision. Out-of-domain sample is routed to the general-purpose LLMs to enhance generalizability. Together, these modules enable ENVUL not only precisely localize vulnerabilities with domain expertise but also retain robustness across all types of vulnerabilities.

We evaluate ENVUL on three high quality real-world datasets: SVEN [26], MegaVul [27], and SafeCoder [28]. Evaluation results show that ENVUL achieves state-of-the-art performance by increasing the top-1 localization accuracy by 22.7%-30.3%, and top-5 results by 8.4%-22.7% over the state-of-the-art tools. Additionally, in experiments focused on generalization, ENVUL significantly boosts top-1 accuracy by 43.6% and 50.0%.

The contributions of this paper are summarized as follows:

- We propose an approach for practical vulnerability localization. Our approach maximizes the potential of LLMs by fully leveraging the domain expertise of task-enhanced fine-tuning and the broad knowledge scope of general LLMs.
- We construct a consolidator to integrate representations of statement-level vulnerability features, and an indicator to capture determinative code elements that directly cause vulnerabilities, thus bolstering the model's domain-specific fine-tuning performance.
- We design a dynamic routing mechanism to distinguish vulnerabilities, enabling fine-tuned models to accurately localize vulnerabilities based on domain expert knowledge. Simultaneously, general models handle scenarios beyond the scope of this domain knowledge to bolster generalizability.

```
static int hq_decode_block(HQContext *c, GetBitContext *gb,
     int16_t block[64], int qsel, int is_chroma, int is_hqa)
      {
   const int32_t *q;
   int val, pos = 1;
   memset(block, 0, 64 * sizeof(*block));
   if (!is_hqa){
      ......
      block[0] = get_sbits(gb, 9) * 64;} //located by SOTA
           tool
   for (;;) {
      ......
      if (pos >= 64) //located by Claude-3.5-sonnet
         break;
      block[ff_zigzag_direct[pos]] =
      (ff_hq_ac_syms[val] * q[pos]) » 12; //located by
      ENVUL
      pos++;}
   return 0;}
```

Fig. 1: The localization results by ENVUL.

- We implement above methods as a tool called ENVUL. Evaluation results on real-world benchmarks show that ENVUL achieves state-of-the-art performance in both accuracy and generalizability, and further experiments validate the effectiveness of each component. We make our implementation open available at https://github.com/TwT23333/ENVUL

II. MOTIVATION

In this section, we present our motivation using several real-world examples.

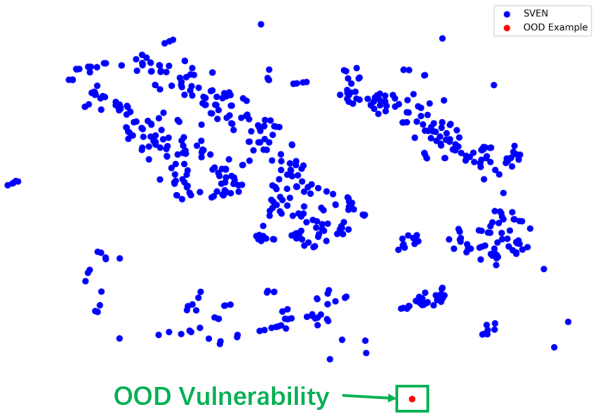
A. Fine-tuning with comprehensive semantic

The challenge of vulnerability localization entails pinpointing vulnerabilities within code statements. It requires analyzing a vulnerable code, denoted as $\mathcal{S}$, and producing output that directly corresponds to, or strongly correlates with, the vulnerabilities. Specifically, given a set of code statements $\mathcal{S} = \{s_1, \ldots, s_n\}$, where s_i denotes the i-th line of code, the objective is to identify a subset $\{s_i, \ldots, s_k\} \subset \mathcal{S}$ causing the vulnerability. Figure 1 depicts a vulnerable code snippet, which is used to decode a block in High-Quality (HQ) video encoding. A negative overflow vulnerability occurs because the multiplication at line 12 may cause integer overflow. Specifically, the type of `q[pos]` is set to `int32_t` instead of `unsigned`, which can result in potential negative values during multiplication operations, thereby leading to integer overflow.

To localize the vulnerable position, it is important to consider the semantics of all the code tokens within a statement. This is because each element in the code, including `ff_hq_ac_syms[val]`, `q[pos]`, their types, and the bitwise shift operation, are closely related to the root cause of the vulnerability. Moreover, to locate this integer overflow issue, it is essential to understand that the vulnerability is caused by `q[pos]`, rather than other code tokens. Therefore, we need to identify this critical information within the statement. Unfortunately, the existing SOTA tuning-based and prompt-based methods, LLMAO [19] and Claude-3.5-sonnet fail to localize the vulnerability. Specifically, LLMAO determines

```
static rt_ssize_t frame_send(struct rt_link_frame *frame){
   rt_size_t length = 0;
   rt_uint8_t *data = RT_NULL;
   rt_memset(rt_link_scb->sendbuffer, 0, sizeof(rt_link_scb
        ->sendbuffer));
   data = rt_link_scb->sendbuffer;
   length = RT_LINK_HEAD_LENGTH;
   length += frame->data_len;
   frame->head.length = frame->data_len;//located by tuned
        model
   rt_memcpy(data, &frame->head, RT_LINK_HEAD_LENGTH);
   data = data + RT_LINK_HEAD_LENGTH;
   if (frame->attribute == RT_LINK_SHORT_DATA_FRAME || frame
        ->attribute == RT_LINK_LONG_DATA_FRAME) {
      rt_memcpy(data, frame->real_data,
      frame->data_len); //located by Claude-3.5-sonnet
      data = data + frame->data_len;}
   return rt_link_hw_send(rt_link_scb->sendbuffer, length);}
```

(1) The localization results of unfamiliar example



(2) Distribution of unfamiliar example and SVEN dataset

Fig. 2: Examples of localization and unfamiliar cases

line 7 to be vulnerable, and Claude-3.5-sonnet incorrectly assumes that the condition check in line 10 is improper.

The causes of mislocalization can be attributed to two aspects: (1) Fine-tune-based approaches lack enhanced statement-level semantic understanding specifically tailored for vulnerability localization tasks. For example, LLMAO considers only newline tokens as the semantic features of each statement, but these account for only a minimal portion of the overall statement's semantics. (2) Prompt-based methods, while leveraging a general LLM, lack specialized vulnerability domain knowledge, leading to errorous results.

In contrast, ENVUL takes into account the deep semantics of all code token with a statement, which enable it to learn the more comprehensive statement-level semantics, i.e., the dependency between `q[pos]` at line 12 and `q`'s definition at line 2. Then, ENVUL highlights `q[pos]` as the determinative element of the statement building upon, and considers it as the key information. Therefore, ENVUL can consider the information of the entire statement comprehensively and ultimately localize the correct vulnerable statement.

B. Wisely choose tuning and prompting

The fine-tuned model performs exceptionally well on familiar data. However, what will happen when it encounters a vulnerability it has never seen before?

Figure 2 (1) displays a recent vulnerability in the RT-Thread project, designated as CVE-2024-25395. The flaw occurs at line 12, where a potential buffer overflow arises when `frame->data_len>1020`. We first train our enhanced fine-tuned model on the SVEN dataset and then use it to locate the vulnerability. However, it still fails to pinpoint the exact location of the vulnerability. To explain this, we first visualize the distribution of the vulnerability data alongside the data in the SVEN dataset, as shown in figure 2 (2). It is evident that the vulnerability lies outside the distribution of the training dataset. In other words, the fine-tuned model has never encountered this type of data before, making it unable to accurately locate the unfamiliar vulnerabilities. However, although Claude-3.5-sonnet previously produced incorrect localization results in the first example, it can still identify this vulnerability by leveraging its extensive knowledge of code. Therefore, relying only on fine-tuned model or a general LLM is insufficient to handle all scenarios effectively.

To address this issue, ENVUL adopts a dynamic domain routing mechanism to effectively route different types of vulnerabilities. For vulnerabilities within the knowledge scope of the dataset, it directs them to the fine-tuned model for localization, maximizing the use of domain-specific knowledge. For unfamiliar vulnerabilities, it leverages a prompt-based approach to utilize general LLM, taking advantage of its extensive code knowledge. This complementary strategy mitigates the risk of incorrect localization by the fine-tuned model and enhances generalization of the tool, making ENVUL a practical vulnerability localization expert.

III. METHODOLOGY

In this section, we present the core idea of ENVUL and provide a detailed explanation of the proposed approach for vulnerability localization, which consists of two components: 1) task-enhanced fine-tuning and 2) dynamic routing mechanism.

Figure 3 shows the overall framework of the proposed approach. ENVUL works as follows: 1) In the task-enhanced fine-tuning phase, consolidator integrates all code element information of a single statement, while indicator identifies key code elements determining whether the statement contains a vulnerability. 2) During the inference phase, the dynamic domain routing mechanism evaluates whether the fine-tuned model is familiar with the vulnerability. This is achieved by analyzing the representations of the training dataset with joint-representation similarity analysis. Based on this evaluation, the mechanism guides ENVUL to either take advantage of the task-enhanced fine-tuned model for localization or fall back on a general LLM with its broad code knowledge, ensuring optimal accuracy and robustness.

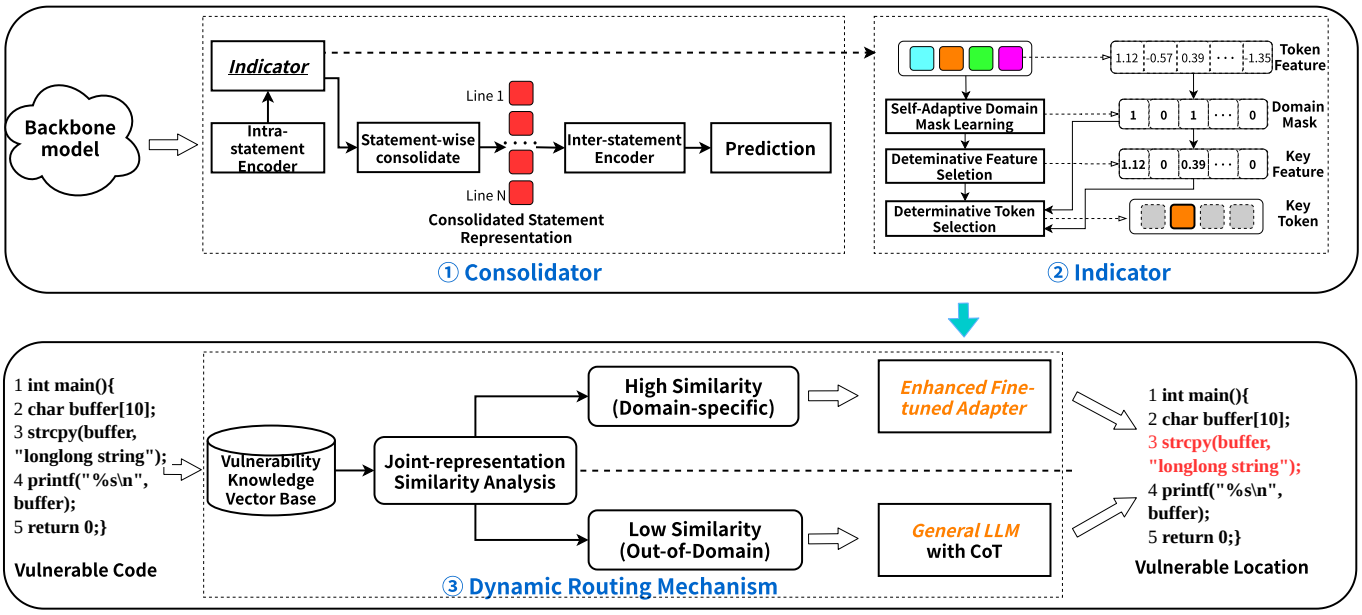


Fig. 3: Overall framework of ENVUL

A. Task-Enhanced Fine-tuning

This section describes how ENVUL enhances performance of fine-tuning by using Consolidator and Indicator designed for the vulnerability localization task.

Data Preprocessing. Given a function $\mathcal{S}$ with N statements of line level, ENVUL tokenizes and processes it through an LLM, obtaining hidden states $L \in \mathbb{R}^{l \times n \times d}$, where l, n, and d represent the number of layers, tokens, and embedding dimensions, respectively. Following insights from prior research [29], we select hidden states from the penultimate layer $\hat{L} \in \mathbb{R}^{n \times d}$ due to their rich semantic representation.

To map tokens to their corresponding statements, we utilize newline tokens as delimiters. Unlike previous methods that capture only a single token per line, ENVUL retains up to $p = 64$ token embeddings per statement. This choice covers 96% of real-world cases observed in our dataset. Padding is applied to ensure uniform dimensions across statements, thus optimizing computational efficiency. Consequently, we obtain statement-level representations as:

$$S = [s_1, s_2, \ldots, s_N], \quad s_i \in \mathbb{R}^{p \times d}, \quad \forall i \in \{1, \ldots, N\}$$

These representations serve as the basis for further semantic enrichment via the consolidator and indicator modules, which extract deeper, task-specific insights from each statement.

Consolidator. After acquiring S, ENVUL aggregates code features at the statement level. To effectively capture contextual dependencies among code elements, we design a context-aware module using a bidirectional attention mechanism. Specifically, token embeddings s_i are first processed by a bidirectional attention layer and subsequently refined by an indicator module (Section III-A), formulated as: $\hat{h}_i = f_{\theta_l}(h_i) = f_{\theta_l}(f_{\theta_t}(s_i))$, where θ_t and θ_l denote parameters of the attention and indicator layers, respectively, and f represents the corresponding transformation functions. After encoding, ENVUL applies average pooling: $g_i = \frac{1}{p}\sum_{j=1}^{p} \hat{h}_i^j$ thus deriving aggregated statement-level representations:

$$G = [g_1, g_2, \ldots, g_N], \quad g_i \in \mathbb{R}^{1 \times d}, \quad \forall i \in \{1, \ldots, N\}.$$

Since vulnerabilities often involve interactions spanning multiple statements, we introduce an inter-statement encoder based on a lightweight Transformer architecture [30] to model cross-statement influences: $\hat{G} = \text{Transformer}(G)$.

We adopt this simplified Transformer structure for two primary reasons. First, given the limited vulnerability-labeled data, a compact model mitigates the risk of overfitting. Second, its lightweight design significantly enhances computational efficiency during the fine-tuning stage.

Indicator. While the aggregation module integrates statement-level global semantics, indicator pinpoints crucial vulnerability patterns by reducing irrelevant code elements that hinder accurate vulnerability localization. The indicator first applies a self-adaptive domain mask on tokens' hidden state to select the most determinative vulnerable feature. Then, it identifies the determinative vulnerable tokens based on the computed weights through attention rectification.

a) Determinative Vulnerable Feature Selection

Parametrically, code language model learns patterns between numerical values to identify features that cause vulnerabilities. However, due to the inherent complexity of code, such as the same identifiers with ambiguous meanings, the model tends to learn many spurious features. This results in output vectors containing numerous redundant values. Therefore, we design a self-adaptive domain mask to identify and prune these spurious features. For the hidden state of a token s_i^k, i.e., k-th token in the i-th line, it is first passed to a learnable self-

adaptive mask $\mathbf{m_f}$ to filter out spurious features while retaining determinative features. The learnable mask $\mathbf{m_f} = \mathbf{w} \odot \mathbf{q}$ consists of binary values, where $\mathbf{q}$ is the pruning template, obtained through a unit step function $\mathbf{q} = step(|h_i^k| - \mathbf{t})$, in which $step(v) = \begin{cases} 0 & \text{if } v < 0 \\ 1 & \text{if } v \geq 0 \end{cases}$ and $\mathbf{t} \in \mathbb{R}^{q \times d}$ is trainable threshold vector for pruning. Moreover, a set of trainable weights $\mathbf{w}$ is used to endow $\mathbf{m_f}$ with adaptive learning capability, allowing it to learn the valid vulnerability features from the training set. By performing element-wise multiplication $h_i^k = s_i^k \odot \mathbf{m_f}$, the zero elements of $\mathbf{m_f}$ effectively eliminate the spurious features in the token embeddings s_i^k. In contrast, the non-zero elements of $\mathbf{m}$ highlight the key features.

b) Determinative Vulnerable Token Selection

After pruning the spurious features of each code token in the previous step, this step aims to eliminate irrelevant code elements and highlight the elements associated with vulnerabilities. For instance, in '`sprintf(buffer, "Number: %d", number);`', the actual cause of the vulnerability is the size of `number` exceeding the memory length of `buffer`. However, during the learning process, the model may incorrectly identify all instances of `sprintf` as vulnerabilities due to its higher frequency of occurrence. Therefore, it is necessary to suppress code tokens that are not related to the vulnerability. For the feature representation h_i obtained in the previous step, we use it as the key vector and the learned mask vector $\mathbf{m_t}$ as the query vector to compute the attention scores $score_i = \mathbf{m_t} h_i^T$, where $\mathbf{m_t}$ is obtained through learning determinative features. The score represents the relationship between each token and the mask, indicating the degree of each token's association with the vulnerability-specific knowledge. Note that we set only one mask for each vulnerability, which represents the domain-specific features.These scores represent the contribution of each token in the statement to the vulnerability. We multiply the score by the features: $\hat{h}_i = score \cdot h_i$. Here, we obtain the token representations most relevant to the vulnerability, i.e., highlighted tokens.

Fine-Tuning Process. During the fine-tuning stage, we freeze the parameters of the backbone model and fine-tune trainable Consolidator and Indicator as an adapter. In statement-level vulnerability localization, the number of statements with vulnerabilities is much smaller than that of normal statements. Hence, we train ENVUL using FocalLoss. FocalLoss helps alleviate the class imbalance issue caused by this phenomenon, accelerating model training and enhancing generalization capability. Specifically, focal loss is defined as follows:

$$FocalLoss(p_t) = -\alpha_t (1 - p_t)^{\gamma} \log(p_t)$$

in which p_t is the model's prediction probability for the correct label. α_t balances class importance, and γ, the focusing parameter, reduces weight for easy samples and increases it for difficult ones. We adopt CodeLlama-7B as our backbone model due to its moderate scale for practical deployment and strong performance on code related tasks.

B. Dynamic Domain Routing for Practical Localization

In this section, we present the algorithm of the dynamic domain routing mechanism and demonstrate how to use it to enhance vulnerability localization.

Algorithm 1: Dynamic Domain Routing Mechanism

```
/* Training Phase                                              */
```
1 **Input:** Backbone model feature and Tuned model feature from vulnerable statements $S = \{s_1, s_2, \ldots, s_n\}$; $G = [g_1, g_2, \ldots, g_N], g_i \in \mathbb{R}^{1 \times d}, \forall i \in \{1, \ldots, N\}$;
2 $ID_{bone} \leftarrow \{\}$; $ID_{ft} \leftarrow \{\}$;
3 **for** $i = 1$ *to* n **do**
4 $\quad \bar{s}_i = \frac{1}{p} \sum_{j=1}^{p} s_i^j$;
5 $\quad \bar{s}_i^{norm} \leftarrow \bar{s}_i / s \|\bar{s}_i\|_2$;
6 $\quad ID_{bone}[i] \leftarrow \bar{s}_i^{norm}$;
7 $\quad g_i^{norm} \leftarrow g_i / \|g_i\|_2$;
8 $\quad ID_{ft}[i] \leftarrow g_i^{norm}$;

```
/* Inference Phase                                             */
```
9 **Input:** Backbone model feature and Tuned model feature of the statement with the highest probability of being vulnerable s_{test}, g_{test}; In-distribution dataset ID_{bone}, ID_{ft}.
10 $\bar{s}_{test} = \frac{1}{p} \sum_{j=1}^{p} s_{test}^j$;
11 $\bar{s}_{test}^{norm} \leftarrow \bar{s}_{test} / \|\bar{s}_{test}\|_2$;
12 $g_{test}^{norm} \leftarrow g_{test} / \|g_{test}\|_2$;
13 $D_{bone} \leftarrow \{\}; D_{ft} \leftarrow \{\}$;
14 **for** $i = 1$ *to* n **do**
15 $\quad id_i^{bone} \leftarrow ID_{bone}^{(i)}$; $id_i^{ft} \leftarrow ID_{ft}^{(i)}$;
16 $\quad d_i^{bone} = \|\bar{s}_{test}^{norm} - id_i^{bone}\|_2$;
17 $\quad d_i^{ft} = \|g_{test}^{norm} - id_i^{ft}\|_2$;
18 $\quad D_{bone} \leftarrow d_i^{bone}; D_{ft} \leftarrow d_i^{ft}$;
19 $D_{bone} \leftarrow \mathtt{Sort}(D_{bone}); D_{ft} \leftarrow \mathtt{Sort}(D_{ft})$;
20 **Output:**The decision whether the k-th smallest distance exceeds the threshold λ, i.e., output 1 if $D_{bone}^{(k)} \geq \lambda \wedge D_{ft}^{(k)} \geq \lambda$, otherwise 0.

Algorithm. The goal of the dynamic domain routing mechanism is to identify the code that falls outside the knowledge scope of the fine-tuned model, hence avoiding it producing false positives for unknown vulnerabilities. Algorithm 1 details our dynamic domain routing mechanism. We implement it based on joint-representation similarity analysis. Specifically, we perform routing by jointly leveraging features from the backbone model's penultimate layer and statement-level representations from the fine-tuned model(line 1). The outputs produced by the backbone model contain broader code semantic information due to its nature of pre-training on a large corpus of code, while the features of fine-tuned model encapsulate more specific vulnerability characteristics. In training phase, ENVUL ascertains the distribution of the training data (line 2), i.e., known data, which will be utilized in subsequent domain analysis. For every statement representation s_i produced by

LLM, ENVUL initially computes the average of its token features, yielding $\bar{s}$ (line 5). This averaging process is designed to generate a vector mirroring the shape of g_i while retaining maximal information from s_i. Subsequently, ENVUL proceeds to normalize each feature vector of s_i. The resultant normalized vectors serve as the basis for computing similarity between training and testing data.

In the inference phase, ENVUL utilizes the K-nearest neighbors algorithm to determine whether a given code snippet is out-of-domain. Specifically, ENVUL extracts features from the statement most confidently identified as vulnerable (line 9). If this feature significantly deviates from knowledge of the training set, it indicates an unfamiliar vulnerability. ENVUL measures similarity between this vector and domain-specific data by L2 distance(lines 15-19), classifying the sample as unfamiliar vulnerability if the distance to the k-th nearest vector exceeds threshold λ (line 20).

Pratical Vulnerability Localization. As described in section 20, we determine whether a vulnerability sample falls outside the fine-tuned model's knowledge boundary by calculating whether its distance from the training set exceeds a predefined threshold. Then, ENVUL dynamically routes the vulnerabilities based on the detection results.

For domain-specific vulnerabilities, since they fall within the knowledge scope of our enhanced fine-tuning model, we directly use it to localize vulnerabilities. ENVUL outputs a probability of containing a vulnerability for each statement, and then we sort these probabilities in descending order, selecting the top-N statements to identify the most likely vulnerable positions. For out-of-domain instances, which are unknown by fine-tuned model, ENVUL will hand them over to general LLM with chain-of-thought prompt for further localization.

> **Prompt:** Analyze the following code for security vulnerabilities and provide your findings in a structured JSON format. The response must contain: `"Functionality"` (a concise description of what the code does) and `"Vulnerabilities"` (an array of objects, each containing `"line"` (the vulnerable line number), `"suspicious"` (rated high-/medium/low) and `"description"` (what makes it vulnerable). Identify up to five most critical issues, explaining your reasoning for each.

Fig. 4: Simplified Chain-of-Thought Prompt

IV. EXPERIMENT

In this section, we evaluate ENVUL's performance on vulnerability localization. The goal of our experiment is to answer the following research questions:

- **RQ1:** Compared to state-of-the-art approaches, how effective is ENVUL in localizing vulnerabilities?
- **RQ2:** How robust is ENVUL in domain-shift vulnerability localization scenario?
- **RQ3:** What are contributions of each component to ENVUL's effectiveness?
- **RQ4:** How is ENVUL's adaptability to different general LLMs?
- **RQ5:** How is ENVUL's real-world performance?

A. Configuration

We set the learning rate to 1e-4, with a data dimension of 4096, and fine-tune ENVUL for 30 epochs, selecting the checkpoint with the best performance on the validation set. We set the model's sampling temperature to 0, ensuring that the model produces the same localization results for the same vulnerability each time. Due to the varying number of lines N per function, the count of statement-level features S obtained also varies, leading to a dynamic batch size.

All the experiments are conducted on a device with 64 cores of 2.3GHz CPU, 128GB RAM, NVIDIA Ampere A100 GPUs, and 40 GB memory. The operating system is Ubuntu 20.04.

B. Dataset and Metrics

To investigate ENVUL's effectiveness on real-world vulnerability localization, we select three real-world vulnerability datasets.

- A multilingual dataset used by SVEN [26] is collected from real-world vulnerability projects and datasets. Each vulnerability is certified by humans to ensure correctness. All vulnerability locations are manually analyzed by experts to identify their root cause. We choose to use SVEN because it has the highest vulnerability statement labeling accuracy among the commonly used vulnerability datasets [31]. It has 803 manually inspected samples.
- MegaVul [27] is a benchmark curated from several Git web hosting services. It filters high-quality vulnerabilities through multiple heuristic methods and utilizes Abstract Syntax Tree and Program Dependency Graph analysis to extract vulnerability-related functions. The dataset contains 169 unique CWE IDs and 8254 CVE IDs.
- Safecoder [28] is a high quality vulnerability dataset that contains 16 CWE types. It is crawled from over 145 million commits from public GitHub projects, and each vulnerability and corresponding location are validated by both manual inspection and static analysis tools. It contains 1211 human-checked vulnerabilities.

According to the study of [31], BigVul [32], Devign [33], and CVEfixes [34] have high number of noises or very low labeling accuracy. Therefore, we choose not to use these datasets.

We employ ten-fold cross-validation to evaluate the performance of ENVUL and the baselines, following the same setting of previous defect localization tools [19], [35]. Note that there is a minor distribution overlap among the three datasets, which will be discussed and tested in RQ2.

In practical applications, developers typically focus on the top results from tools. Hence, we adopt TOP-N as our experimental evaluation metric, aligning with what state-of-the-art

tools commonly use [19], [36]. TOP-N measures a model's localization capability by calculating if at least one of the model's top N predicted lines of code matches the ground truth. In this paper, we select top-1, top-3, and top-5 as our evaluation metrics.

C. RQ1.Effectiveness

In this research question, we evaluate the effectiveness of ENVUL on vulnerability localization.

Setting. To evaluate the performance of the ENVUL, we chose UnixCoder [37], LineVD [1], LineVul [18], Claude3.5 sonnet-CoT, DeepSeek-R1 [38], Codellama (fine-tune and CoT) and LLAMO [19] as baselines.

- **UnixCoder** is a code language model specifically designed for understanding code. It leverages Abstract-Syntax-Tree(AST) based pre-training to improve its ability to capture semantic representations.
- **LineVD** is an advanced vulnerability localization tool that leverages graph neural network to learn representation from Program Dependence Graph(PDG).
- **LineVul** is a widely used tool that utilizes attention mechanisms within the BERT architecture [39] to predict line-level vulnerability.
- **Claude3.5 sonnet** [40] is currently one of the most advanced large language models for program understanding. Since it is a closed-source model, we cannot access its output feature vectors. Therefore, we utilize its API with a chain-of-thought prompt.
- **DeepSeek-R1** is a reasoning model enhanced through reinforcement learning, which is excellent in coding tasks. We also use its API for evaluation.
- For **CodeLlama-7B**, we adopt two experimental settings: fine-tune and CoT. For the former, we extract only the newline token's hidden state as its feature representation, then fine-tune a single-layer fully connected network as the prediction model to output the probability of the statement being vulnerable. For CoT, we directly use the pre-trained model and employ the same prompt as described in section 20. This aims to evaluate the effectiveness of our enhanced fine-tuning model and the zero-shot handling capability of CodeLlama as a general LLM.
- **LLMAO** is the state-of-the-art vulnerability localization tool, which employs a large language model and freezes its weight, and fine-tunes an additional adapter to predict vulnerability with the LLM feature of the newline token produced before. The original paper for LLMAO uses CodeGen-16B as the backbone, and we reimplement it with CodeLlama-7B to maintain the same settings as our tool.

Results. Tabel I shows the results of baselines and ENVUL. Our tool demonstrates superior performance over all other tools across three datasets. Furthermore, compared with state-of-the-art tools, our model outperforms LLMAO in top-1 accuracy on the SVEN, MegaVul, and SafeCoder datasets, with increases of 30.32%, 22.71%, and 25.97% respectively. Additionally, the top-3 accuracy rates increase by 25.01%, 12.44%, and 13.72%, respectively, while the top-5 accuracy rates improve by 22.70%, 13.72%, and 8.40%. The experimental results can be attributed to the deep semantic representation of code provided by the consolidator and indicator. Moreover, our domain routing mechanism enables ENVUL to handle unfamiliar data, thereby improving the performance of vulnerability localization.

Although ENVUL achieves progressive results, the performance of our tool is inconsistent across different datasets. After analyzing the datasets, we identify the following reasons. First, the SVEN dataset, which is a multilingual dataset and each vulnerability is manually certified by human experts, contains the most diverse types and languages of vulnerabilities and has more complex features. Hence, our tool performs moderately on this dataset. Second, compared to SVEN, the SafeCoder dataset is verified by static analysis tools(i.e., CodeQL) and humans. Static analysis tools often rely on manually crafted rules, which tend to result in a consistent range of detected vulnerabilities, making it easier for the model to learn their characteristics, thus our tool shows the best performance on this dataset. Last, MegaVul design an automated crawling method with multiple heuristics. Although this approach improves label accuracy, it still contains incorrectly labeled vulnerabilities. Therefore, it is challenging for ENVUL to learn the features correctly, resulting in performance degradation on this dataset.

Moreover, to evaluate performance in practical vulnerability localization scenarios, we evaluate ENVUL's accuracy in localizing the Top-25 most dangerous CWEs with Top-5 metric. We select vulnerability types that appear multiple times in the datasets. As shown in Table II, ENVUL's accuracy in localizing the Top-25 most dangerous CWEs ranges from 76% to 100%, with an average accuracy reaching 91%. This validates the practical utility of our tool in real-world applications.

> RQ1 Answer: Benefiting from *Consolidator* and *Indicator* for enhancing the deep semantic information during fine-tuning and *Dynamic Routing* for improving out-of-domain capability, ENVUL achieves a 22.71% - 30.32% enhancement on Top-1 over the SOTA baseline in real-world datasets.

D. RQ2.Robustness Under Domain-shift Scenario

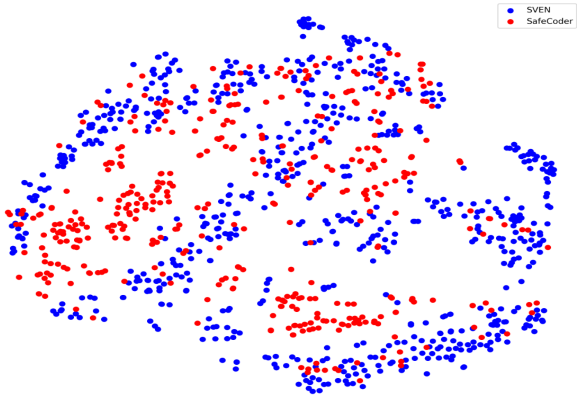
In this research question, we first clarify the differences in dataset distributions, and then, based on this understanding, we evaluate ENVUL's generalization capabilities. Additionally, we test the ability of the routing mechanism to detect out-of-domain samples and its contribution to improving the accuracy of vulnerability localization.

1) Localization Performance Under Domain-shift

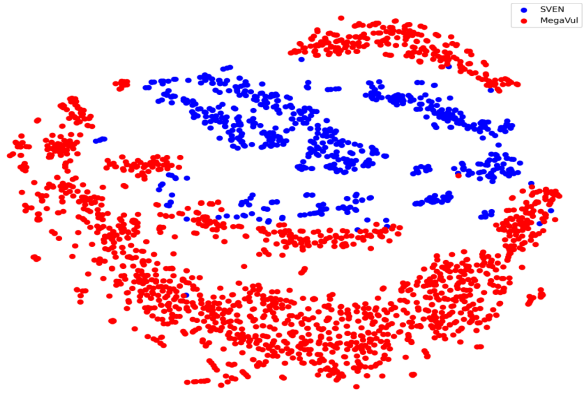
Dataset shift and Setting. Before addressing this research question, we first examine the distribution differences between datasets. We use t-SNE [41] for data visualization. In Figures 5, red represents the SVEN training set, while blue and red represent the SafeCoder and MegaVul test sets, respectively. The figures show partial overlap between SVEN and Safe-

TABLE I: (RQ1) Performance of ENVUL of real-world datasets and comparison with the baseline

Tool		SVEN			MegaVul			SafeCoder		
		TOP-1	TOP-3	TOP-5	TOP-1	TOP-3	TOP-5	TOP-1	TOP-3	TOP-5
CoT	CodeLlama$_{CoT}$	26.46%	36.76%	39.70%	28.02%	43.84%	49.07%	42.86%	54.75%	64.24%
	DeepSeek-R1	37.29%	48.85%	56.81%	40.35%	55.79%	65.88%	48.39%	62.18%	70.52%
	Claude3.5 sonnet	33.81%	42.65%	54.41%	33.25%	54.30%	61.39%	47.56%	59.45%	64.66%
GNN	LineVD	31.09%	41.85%	56.43%	34.25%	51.81%	60.52%	45.09%	56.65%	63.83%
AST	UnixCoder	38.24%	52.93%	59.63%	47.32%	57.78%	71.86%	49.96%	61.85%	69.03%
FT	LineVul	44.11%	55.88%	61.75%	45.58%	56.04%	68.37%	52.35%	59.45%	66.64%
	CodeLlama$_{FT}$	36.76%	44.11%	51.47%	38.48%	49.07%	59.53%	47.56%	57.06%	64.85%
	LLMAO	48.52%	58.82%	64.71%	54.30%	70.11%	77.09%	64.24%	71.35%	85.63%
FT&CoT	ENVUL	**63.23%**	**73.53%**	**79.40%**	**66.63%**	**78.83%**	**87.67%**	**80.92%**	**90.42%**	**92.82%**



(a) Near-shift: SVEN vs SafeCoder



(b) Far-shift: SVEN vs MegaVul

Fig. 5: Differences in distribution among various real-world datasets

Rank	CWE Type Name	Acc
1	CWE-079 Cross-site Scripting	87%
2	CWE-787 Out-of-bounds Write	97%
3	CWE-089 SQL Injection	76%
4	CWE-020 Improper Input Validation	90%
5	CWE-022 Path Traversal	87%
6	CWE-125 Out-of-bounds Read	93%
8	CWE-416 Use After Free	100%
13	CWE-077 Command Injection	85%
17	CWE-200 Exposure of Sensitive Info	95%
20	CWE-119 Buffer Overflow	91%
23	CWE-190 Integer Overflow	95%
TOTAL		**91%**

TABLE II: CWE TPR Proportion Table

Coder, with some shifts, whereas SVEN and MegaVul have almost no overlap. Previous Figure 5 examined the situation from the perspective of data scale. Moreover, from a dataset collection viewpoint, since SVEN and SafeCoder may utilize similar projects or datasets during their construction, their distributions are quite similar. In contrast, the data collection process of MegaVul is automated and the crawled projects are different. Therefore, it differs significantly from SVEN.

Consequently, we categorize SafeCoder as a near-shift dataset, where a segment of the distribution shifts relative to the source set, and MegaVul as a far-shift dataset, where the majority of the data has shifted, displaying minimal overlap with the source set. Thus, we aim to test ENVUL's generalization ability on programs to simulate scenarios in which the training data differs significantly from real-world application contexts. We train ENVUL using SVEN and then evaluate its effectiveness on the near-shift and far-shift datasets.

TABLE III: (RQ2) ENVUL's Capability on Shifted Datasets

Setting / Tool	Near-shift			Far-shift		
	TOP-1	TOP-3	TOP-5	TOP-1	TOP-3	TOP-5
LLMAO	35.71%	59.52%	73.81%	31.58%	52.63%	61.40%
ENVUL	**51.26%**	**71.96%**	**83.74%**	**47.37%**	**63.16%**	**69.63%**

Results. Table III shows the experimental results on the near-shift dataset and far-shift dataset. For the near-shift dataset, ENVUL achieved improvements of 43.55%, 20.90%,

and 13.45% in top-1, top-3, and top-5 metrics, respectively, compared to SOTA tool LLMAO, showing stable performance similar to single-dataset settings in RQ1. On the far-shift dataset, increases of 50.01%, 20.11%, and 13.40% were noted in these metrics. Moreover, as shown in Figure III, ENVUL also achieves significantly improved performance compared to general LLMs, demonstrating the effectiveness of our routing mechanism. These results demonstrate ENVUL's capability to learn complex semantic features of programs and generalize across different codes, highlighting our tool's generalization ability.

2) Routing Performance Under Domain-shift

Setting. For testing effectiveness of routing mechanism, we evaluate how many out-of-domain samples our module could detect under near-shift and far-shift conditions. We computed the accuracy of domain discrimination as follows:

$$Accuracy = \frac{Number\ of\ correct\ out\text{-}of\text{-}domain\ sample}{Total\ number\ of\ predictions}$$

Since there is currently no standardized criterion for distinguishing domain-specific data, we treat codes that the model accurately locates vulnerabilities in as domain-specific samples.

Results. Table IV presents our experimental outcomes. In the original setting, the dynamic routing mechanism detected 101 out-of-domain samples and route them to general LLM, but also mistakenly removes 14 domain-specific samples, with an accuracy of 88.71%. For near-shift, it routes 68 out-of-domain samples, achieving an accuracy of 94.44%. In far-shift, due to minimal dataset overlap, our module performs best, correctly routing 91 samples, with an accuracy of 97.85%. These results align with our data visualization in Figure 5, indicating higher accuracy with greater distribution divergence.

TABLE IV: (RQ2) Performance of Dynamic Routing

Dataset	Correct	False	Acc.
Original	101	15	87.07%
Near-shift	68	4	94.44%
Far-shift	91	2	97.85%

RQ2 Answer: ENVUL exhibits substantial ability in dynamic routing mechanism and demonstrates superior localization performance compared to the SOTA baseline across domain shifts, showcasing its robust generalization capabilities.

E. RQ3.Ablation study

We conduct an ablation study to assess the contribution of each component in ENVUL. The result is shown in Table V.

Setting. We evaluate ENVUL with the following setting.

- *Plain LLM.* First, we remove all components to test the effectiveness of directly localizing vulnerabilities with the pre-trained. We utilize CodeLlama-7B in this experiment, which is base LLM in ENVUL. We follow the setup described in RQ1.
- *w/o Consolidator.* To measure the impact of the consolidator, we remove it and average tokens' hidden state of each statement to simulate the consolidator without the component.
- *w/o Indicator.* To prove the effectiveness of the indicator, we directly remove it for making subsequent predictions.
- *only Fine-Tune.* To test the performance of the routing mechanism, we removed it and the subsequent general LLM, using only the enhanced fine-tuned model to handle all vulnerabilities.

Results. Table V Row 1 and Row 2 show the results, only using prompt or regular fine-tuning results in a significant performance drop. In contrast, ENVUL's performance on top-1 localization improved by 138.91%, 137.51%, and 88.87% across three datasets, which demonstrates that the usefulness of ENVUL is not solely due to the LLM's understanding of code semantics but significantly to the contributions of other components.

The results are displayed in Table V Row 2. Compared to removing consolidator, ENVUL shows significant improvement on all three datasets. Specifically, the top-1 localization accuracy increases by 7.51%, 15.17%, and 9.67%, respectively. Similarly, the top-3 and top-5 localization accuracy also rise considerably, indicating the substantial contribution of the consolidator to the overall performance. Note that the localization performance of simply averaging hidden states of a statement is even worse than LLMAO, which just extracts one hidden state per line. This further validates that without appropriate consolidator of statement-level information prevents the representation of viable features.

Table V Row 3 presents the results. With the enhancement from the indicator architecture, the localization performance dropped by 2.40%, 5.56%, and 3.03% on top-1, respectively. This reflects the indicator's strength in recognizing determinative vulnerability features and making ENVUL more robust.

As shown in Table V, compared to only fine-tuning, ENVUL achieves improvements in the top-1 metric by 10.27%, 8.58%, and 6.25%, in the top-3 metric by 8.69%, 7.15%, and 2.70%, and top-5 metric by 10.20%, 11.11%, and 1.54%. This demonstrates that our routing mechanism complements the fine-tuned model and general LLM, thereby enhancing the overall capability of the tool.

We also investigate ENVUL's hyperparameter sensitivity, i.e., λ. In real-world scenarios, prior access to out-of-domain vulnerabilities is unavailable, rendering the learning of adaptive thresholds impossible. Therefore, we establish a decision boundary by setting a lambda threshold(λ=0.97) to effectively distinguishing out-of-domain vulnerabilities while preventing overfitting to domain-specific samples. We conduct an sensitivity analysis, adopting the settings from RQ2 and using Top-3 accuracy as the evaluation metric. As shown in Table VI, ENVul performs optimally with λ=0.97. Users can set the threahold according their usage scenarios.

TABLE V: (RQ3) Contribution of each component of ENVUL

Module \ Test Set	SVEN			MegaVul			SafeCoder		
	TOP-1	TOP-3	TOP-5	TOP-1	TOP-3	TOP-5	TOP-1	TOP-3	TOP-5
CodeLlama$_{CoT}$	26.46%	36.76%	39.70%	28.02%	43.84%	49.07%	42.86%	54.75%	64.24%
CodeLlama$_{FT}$	36.76%	44.11%	51.47%	38.48%	49.07%	59.53%	47.56%	57.06%	64.85%
w/o Consolidator	58.82%	70.59%	75.01%	57.89%	71.93%	80.70%	73.81%	83.33%	85.71%
w/o Indicator	61.76%	70.59%	76.47%	63.16%	75.44%	85.07%	78.57%	87.24%	89.27%
only FT	57.35%	67.65%	72.06%	61.40%	73.68%	78.95%	76.19%	88.10%	91.45%
ENVUL	**63.23%**	**73.53%**	**79.40%**	**66.63%**	**78.83%**	**87.67%**	**80.92%**	**90.42%**	**92.82%**

TABLE VI: Hyperparameter Sensitivity

λ	**0.9**	**0.93**	**0.95**	**0.97**	**0.99**
Near	68.13%	69.20%	69.78%	71.96%	68.04%
Far	61.51%	61.76%	61.93%	63.16%	62.38%

RQ3 Answer: ENVUL's consolidator and indicator effectively enrich statement-level information and enhance the model's learning capabilities, and dynamic routing improves the ENVUL's effectiveness on out-of-domain data. Both complement each other, improving the vulnerability localization performance.

F. RQ4.Adaptability to General LLM

For this research question, we evaluate how effectively ENVUL performs across different general LLMs.

Setting. We maintain the fine-tuned adapter and routing mechanism unchanged and utilize three different general LLMs: CodeLlama-7B(backbone model for fine-tuning), Claude-3.5, and DeepSeek-R1. We follow the setting of RQ1 for training and testing and use Top-1 as metric.

Results. Figure 6 shows the results of our general LLM selection experiments. As the performance of the general LLM improves, ENVUL achieves consistently better results. The adaptability of our approach allows for seamless integration with various general LLMs for effective vulnerability localization tasks. This capability indicates that ENVUL's vulnerability localization performance can be enhanced by leveraging more capable general LLMs, further demonstrating the inherent scalability of our tool.

RQ4 Answer: ENVUL seamlessly integrates with various general LLMs, enabling it to achieve better vulnerability localization as LLM capability increases.

G. RQ5.Real World Performance

We further evaluate ENVUL's real-world performance in this research question.

Setting. To mitigate potential data leakage concerns of LLMs, we manually crawl 327 CVEs from April 2024(The knowledge cut-off date of Claude 3.5 Sonnet) to Feb 2025.

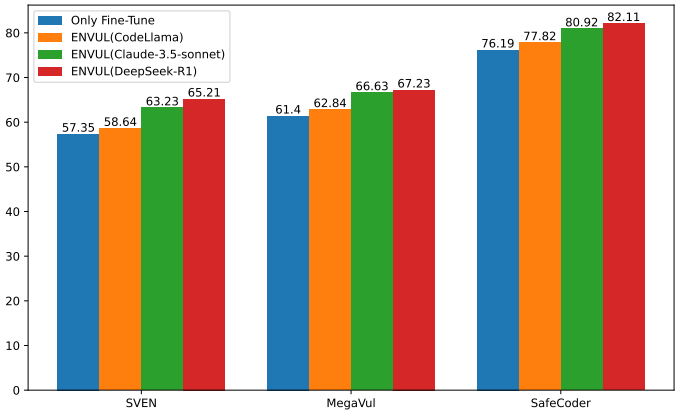


Fig. 6: RQ4. ENVUL's Adaptability to General LLM

Subsequently, we use SVEN as training set and use the crawled unseen CVEs as testing set. Moreover, we chose the PrimeVul [31] dataset to further assess ENVUL's performance due to its real-world nature, top-tier quality, and extensive use within the vulnerability research works [42]–[44]. We obtain 2831 patches through the commit links provided by PrimeVul, and conduct evaluation following the setting of RQ1.

Results. The results of locating unseen CVEs are displayed in Table VII. Even on previously unseen new vulnerabilities, ENVul still demonstrates stable localization effectiveness.

TABLE VII: Performance on Unseen-CVEs

Model	**Top-1**	**Top-3**	**Top-5**
LLMAO	34.86%	44.65%	56.88%
ENVul	44.34%	60.55%	68.50%

Furthermore, ENVul maintains strong performance on PrimeVul as shown in Table VIII. This demonstrates that ENVUL can not only identify vulnerabilities within its knowledge scope through model fine-tuning, but also locate unknown vulnerabilities in real-world scenarios by leveraging the generalization knowledge of a general-purpose LLM.

TABLE VIII: Performance on PrimeVul Dataset

Model	Top-1	Top-3	Top-5
LLMAO	30.74%	41.34%	49.82%
ENVul	38.52%	50.18%	62.90%

RQ4 Answer: By combining the specialized capabilities of fine-tuned models with the generalization knowledge of general models, ENVUL also demonstrates strong performance in locating unknown vulnerabilities in real-world scenarios.

V. THREATS TO VALIDITY

Threats to internal validity. The selection of hyperparameters may affect the performance of our ENVUL. We set the learning rate to a fixed 1e-4 to accelerate convergence, but this might lead to suboptimal results, ultimately causing the localization performance to deviate from the ideal outcome. Due to the complexity and diversity of vulnerability data, our dynamic routing mechanism cannot completely eliminate the influence of out-of-domain data, which means fine-tuned models will still encounter many unfamiliar vulnerabilities, leading to incorrect localization results.

Threats to external validity. To mitigate the generalization challenge of ENVUL, we employ three datasets in experiments, which contain extensive real-world vulnerability data. We evaluate the baseline methods and ENVUL fairly across these three datasets. In future work, we will explore the performance of ENVUL on other datasets.

VI. RELATED WORK

We categorize related work into three parts: vulnerability detection, vulnerability localization (relevant to the purpose of ENVUL), and out-of-distribution detection (relevant to the method proposed in ENVUL).

Vulnerability detection. Traditional static vulnerability detection efforts [3], [13], [14], [45]–[51], which necessitate intense manual labor (e.g., feature definition), often lead to a high false negative rate [2]. In response, a series of deep learning-based approaches [2], [4], [33] have been introduced. Devign [33] utilizes graph neural networks to detect code vulnerabilities. VulBG [52] adopts a Behavior Graph Model to leverage global contextual information for detecting vulnerabilities. However, these function-level vulnerability detection approaches typically only identify issues at the function level, with a coarse granularity. This limits developers' ability to effectively inspect and interpret the predictions of learning models.

Vulnerability localization. Locating vulnerabilities on the specific lines can effectively reduce the manual effort required for vulnerability detection and enhance the Interpretability of the detection results, hence garnering increasing attention in recent years [1], [7], [18], [19], [53]–[58]. LineVul [18] utilizes attention mechanisms within the BERT [39] architecture for line-level vulnerability prediction. Yang et al. [19] proposed LLMAO, which fine-tunes a small group of bidirectional adapter layers. However, it only utilizes the representation of a single token per line, thereby losing a lot of important information. SoapFL [20] is a LLM-driven standard operating procedure to automatically localize buggy methods. We do not use it for evaluation because SoapFL is suitable for method-level fault localization and requires test cases, while ENVul is a statement-level vulnerability localization tool and does not require test cases. More importantly, due to the discrepancy between real data and training data, these methods often yield seemingly trustworthy but erroneous results when facing undetectable vulnerabilities. Our work, ENVUL, employs OOD detection to identify vulnerabilities outside the expected distribution and utilizes consolidator and indicator to obtain more comprehensive code representation.

VII. CONCLUSION

In this paper, we introduced ENVUL, a novel domain adaptation framework for vulnerability localization. By synergizing enhanced task-specific tuning with strategic prompt engineering, ENVUL bridges the gap between specialized domain knowledge and generalizability. Our context Consolidator and semantic Indicator components improve statement-level semantic understanding and vulnerability pattern recognition, while the dynamic routing mechanism intelligently leverages the strengths of both fine-tuned and general-purpose LLMs. Evaluations on real-world vulnerabilities demonstrate ENVUL's superior performance and exceptional generalization capabilities, with 22.7%-30.3% improvements in top-1 accuracy and 43.6%-50% higher accuracy on unfamiliar vulnerability types.

ACKNOWLEDGMENT

This work was supported by the National Key R&D Program of China No 2024YFB4506200 and National Natural Science Foundation of China under Grant No 62202026.

REFERENCES

[1] D. Hin, A. Kan, H. Chen, and M. A. Babar, "Linevd: Statement-level vulnerability detection using graph neural networks," in *Proceedings of the 19th international conference on mining software repositories*, 2022, pp. 596–607.

[2] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "Vuldeepecker: A deep learning-based system for vulnerability detection," *arXiv preprint arXiv:1801.01681*, 2018.

[3] Z. Li, D. Zou, S. Xu, H. Jin, H. Qi, and J. Hu, "Vulpecker: an automated vulnerability detection system based on code similarity analysis," in *Proceedings of the 32nd annual conference on computer security applications*, 2016, pp. 201–213.

[4] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "Sysevr: A framework for using deep learning to detect software vulnerabilities," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2244–2258, 2021.

[5] X. Gao, B. Wang, G. J. Duck, R. Ji, Y. Xiong, and A. Roychoudhury, "Beyond tests: Program vulnerability repair via crash constraint extraction," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 2, pp. 1–27, 2021.

[6] Y. Zhang, X. Gao, G. J. Duck, and A. Roychoudhury, "Program vulnerability repair via inductive inference," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022, pp. 691–702.

[7] Z. Li, D. Zou, S. Xu, Z. Chen, Y. Zhu, and H. Jin, "Vuldeelocator: a deep learning-based fine-grained vulnerability detector," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2821–2837, 2021.

[8] Y. Shen, X. Gao, H. Sun, and Y. Guo, "Understanding vulnerabilities in software supply chains," *Empirical Software Engineering*, vol. 30, no. 1, pp. 1–38, 2025.

[9] Q. Dong, Y. Lin, H. Sun, and X. Gao, "Enhancing automated vulnerability repair through dependency embedding and pattern store," in *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2025, pp. 193–204.

[10] Y. Song, X. Gao, W. Li, W.-N. Chin, and A. Roychoudhury, "Provenfix: Temporal property-guided program repair," *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, pp. 226–248, 2024.

[11] S. Hong, H. Sun, X. Gao, and S. H. Tan, "Investigating and detecting silent bugs in pytorch programs," in *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2024, pp. 272–283.

[12] Z. Fan, X. Gao, M. Mirchev, A. Roychoudhury, and S. H. Tan, "Automated repair of programs from large language models," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1469–1481.

[13] (2024) Clang static analyzer,. [Online]. Available: https://clang-analyzer.llvm.org/scan-build.html

[14] (2024) Infer. [Online]. Available: https://fbinfer.com/

[15] Anthropic, "Introducing computer use, a new claude 3.5 sonnet, and claude 3.5 haiku," Oct. 2024, accessed: 2024-10-26. Available: https://www.anthropic.com/news/3-5-models-and-computer-use.

[16] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford *et al.*, "Gpt-4o system card," *arXiv preprint arXiv:2410.21276*, 2024.

[17] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[18] M. Fu and C. Tantithamthavorn, "Linevul: A transformer-based line-level vulnerability prediction," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 608–620.

[19] A. Z. Yang, C. Le Goues, R. Martins, and V. Hellendoorn, "Large language models for test-free fault localization," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–12.

[20] Y. Qin, S. Wang, Y. Lou, J. Dong, K. Wang, X. Li, and X. Mao, "S oap fl: A standard operating procedure for llm-based method-level fault localization," *IEEE Transactions on Software Engineering*, 2025.

[21] J. Zhang, C. Wang, A. Li, W. Sun, C. Zhang, W. Ma, and Y. Liu, "An empirical study of automated vulnerability localization with large language models," *arXiv preprint arXiv:2404.00287*, 2024.

[22] V. Rawte, A. Sheth, and A. Das, "A survey of hallucination in large foundation models," *arXiv preprint arXiv:2309.05922*, 2023.

[23] N. M. Guerreiro, D. M. Alves, J. Waldendorf, B. Haddow, A. Birch, P. Colombo, and A. F. Martins, "Hallucinations in large multilingual translation models," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 1500–1517, 2023.

[24] X. Guan, Y. Liu, H. Lin, Y. Lu, B. He, X. Han, and L. Sun, "Mitigating large language model hallucinations via autonomous knowledge graph-based retrofitting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 16, 2024, pp. 18 126–18 134.

[25] L. Yang, Y. Song, X. Ren, C. Lyu, Y. Wang, J. Zhuo, L. Liu, J. Wang, J. Foster, and Y. Zhang, "Out-of-distribution generalization in natural language processing: Past, present, and future," in *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[26] J. He and M. Vechev, "Large language models for code: Security hardening and adversarial testing," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 1865–1879.

[27] C. Ni, L. Shen, X. Yang, Y. Zhu, and S. Wang, "Megavul: Ac/c++ vulnerability dataset with comprehensive code representations," in *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. IEEE, 2024, pp. 738–742.

[28] J. He, M. Vero, G. Krasnopolska, and M. Vechev, "Instruction tuning for secure code generation," in *Forty-first International Conference on Machine Learning*.

[29] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning–based text classification: a comprehensive review," *ACM computing surveys (CSUR)*, vol. 54, no. 3, pp. 1–40, 2021.

[30] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[31] Y. Ding, Y. Fu, O. Ibrahim, C. Sitawarin, X. Chen, B. Alomair, D. Wagner, B. Ray, and Y. Chen, "Vulnerability detection with code language models: How far are we?" in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 2024, pp. 469–481.

[32] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "Ac/c++ code vulnerability dataset with code changes and cve summaries," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 508–512.

[33] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[34] G. Bhandari, A. Naseer, and L. Moonen, "Cvefixes: automated collection of vulnerabilities and their fixes from open-source software," in *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2021, pp. 30–39.

[35] Y. Li, S. Wang, and T. Nguyen, "Fault localization with code coverage representation learning," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 661–673.

[36] X. Meng, X. Wang, H. Zhang, H. Sun, and X. Liu, "Improving fault localization and program repair with deep semantic features and transferred knowledge," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1169–1180.

[37] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "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)*, 2022, pp. 7212–7225.

[38] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[39] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "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, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[40] Claude, "Claude 3.5 sonnet," https://www.anthropic.com/news/claude-3-5-sonnet, 2024, accessed: 2024-09-12.

[41] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

[42] G. Team, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.

[43] Y. Guo, C. Patsakis, Q. Hu, Q. Tang, and F. Casino, "Outside the comfort zone: Analysing llm capabilities in software vulnerability detection," in *European symposium on research in computer security*. Springer, 2024, pp. 271–289.

[44] X. Zheng, X. Qian, H. Zhou, S. Yang, Y. He, S. Jana, and L. Cavallaro, "Learning to focus: Context extraction for efficient code vulnerability detection with language models," *arXiv preprint arXiv:2505.17460*, 2025.

[45] J. Viega, J.-T. Bloch, Y. Kohno, and G. McGraw, "Its4: A static vulnerability scanner for c and c++ code," in *Proceedings 16th Annual Computer Security Applications Conference (ACSAC'00)*. IEEE, 2000, pp. 257–267.

[46] Y. Sui and J. Xue, "Svf: interprocedural static value-flow analysis in llvm," in *Proceedings of the 25th international conference on compiler construction*, 2016, pp. 265–266.

[47] F. Yamaguchi, C. Wressnegger, H. Gascon, and K. Rieck, "Chucky: Exposing missing checks in source code for vulnerability discovery," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 499–510.

[48] F. Yamaguchi, A. Maier, H. Gascon, and K. Rieck, "Automatic inference of search patterns for taint-style vulnerabilities," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 797–812.

[49] M. Backes, B. Köpf, and A. Rybalchenko, "Automatic discovery and quantification of information leaks," in *2009 30th IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 141–153.

[50] N. H. Pham, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Detection of recurring software vulnerabilities," in *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*, 2010, pp. 447–456.

[51] S. Kim, S. Woo, H. Lee, and H. Oh, "Vuddy: A scalable approach for vulnerable code clone discovery," in *2017 IEEE symposium on security and privacy (SP)*. IEEE, 2017, pp. 595–614.

[52] B. Yuan, Y. Lu, Y. Fang, Y. Wu, D. Zou, Z. Li, Z. Li, and H. Jin, "Enhancing deep learning-based vulnerability detection by building behavior graph model," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2262–2274.

[53] Y. Li, S. Wang, and T. 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*, 2021, pp. 292–303.

[54] C. Pornprasit and C. K. Tantithamthavorn, "Jitline: A simpler, better, faster, finer-grained just-in-time defect prediction," in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 369–379.

[55] C. Pornprasit and C. Tantithamthavorn, "Deeplinedp: towards a deep learning approach for line-level defect prediction," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 84–98, 2023.

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

[57] V. Nguyen, T. Le, O. De Vel, P. Montague, J. Grundy, and D. Phung, "Information-theoretic source code vulnerability highlighting," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[58] X. Yin, C. Ni, and S. Wang, "Multitask-based evaluation of open-source llm on software vulnerability," *IEEE Transactions on Software Engineering*, 2024.