2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) | 979-8-3503-3066-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/SANER60148.2024.00102

Dual Prompt-Based Few-Shot Learning for Automated Vulnerability Patch Localization

Junwei Zhang[1], Xing Hu[1*], Lingfeng Bao[1], Xin Xia[2], Shanping Li[1]
[1]*The State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou, China*
[2]*Software Engineering Application Technology Lab, Huawei, China*
{jw.zhang, xinghu, lingfengbao}@zju.edu.cn, xin.xia@acm.org, shan@zju.edu.cn

***Abstract*—Vulnerabilities are disclosed with corresponding patches so that users can remediate them in time. However, there are instances where patches are not released with the disclosed vulnerabilities, causing hidden dangers, especially if dependent software remains uninformed about the affected code repository. Hence, it is crucial to automatically locate security patches for disclosed vulnerabilities among a multitude of commits. Despite the promising performance of existing learning-based localization approaches, they still suffer from the following limitations: (1) They cannot perform well in data scarcity scenarios. Most neural models require extensive datasets to capture the semantic correlations between the vulnerability description and code commits, while the number of disclosed vulnerabilities with patches is limited. (2) They struggle to capture the deep semantic correlations between the vulnerability description and code commits due to inherent differences in semantics and characters between code changes and commit messages. It is difficult to use one model to capture the semantic correlations between vulnerability descriptions and code commits.**

To mitigate these two limitations, in this paper, we propose a novel security patch localization approach named PromVPat, which utilizes the dual prompt tuning channel to capture the semantic correlation between vulnerability descriptions and commits, especially in data scarcity (i.e., few-shot) scenarios. We first input the commit message and code changes with the vulnerability description into the prompt generator to generate two new inputs with prompt templates. Then, we adopt a pre-trained language model (i.e., PLM) as the encoder, utilize the prompt tuning method to fine-tune the encoder, and generate two correlation probabilities as the semantic features. In addition, we extract 26 handcrafted features from the vulnerability descriptions and the code commits. Finally, we utilize the attention mechanism to fuse the handcrafted and semantic features, which are fed into the classifier to predict the correlation probability and locate the security patch. To evaluate the performance of PromVPat, we compare it with five baselines on two datasets. Experimental results demonstrate that PromVPat performs best in the security patch localization task, improving the best baseline by 14.42% and 86.57% on two datasets regarding Recall@1. Moreover, PromVPat has proven to be effective even in data scarcity scenarios.

***Index Terms*—Vulnerability Patch Localization, Few-Shot Learning, Pre-trained Language Models**

I. Introduction

Open Source Software (OSS) has transformed the dynamics of contemporary software development, making software more accessible, versatile, and collaborative. However, this public nature also renders OSS a prime target for cyber threats. The upsurge in disclosed OSS vulnerabilities highlights this threat landscape [1]–[3]. Security patches are crucial in addressing these vulnerabilities, often seen as the primary solution. However, as the scale and complexity of OSS increase, a concerning trend emerges: security patches are often silently merged into their repositories, absent any explicit link to the documented vulnerabilities they address. If not notified, software dependent on these code repositories could be exposed to concealed risks [4].

Applying the latest version in time is a common and effective practice to mitigate the risk of vulnerability attacks. However, the latest version, which may contain not only security vulnerability fixes but also new features or performance bug fixes, can overwhelm users or administrators [5]. Besides, it is crucial to facilitate the awareness of vulnerability fixes for OSS users so they can understand the root cause and react to the vulnerabilities as early as possible. Thus, it is important to identify the security patch instead of simply updating the software to the latest version. In response to this need, many data-driven techniques have emerged. These techniques aim to frustrate the exploitation of these vulnerable libraries and facilitate the automatic detection of security patches within a vast volume of code commits [6]–[13].

Generally, existing localization approaches can be categorized into two types: (1) matching-based and (2) learning-based approaches. Matching-based methods depend on specific keywords or supplementary information in the vulnerability description. For instance, Kim et al. [6] used the CVE-ID in code commits to locate security patches. Other studies [7], [8] have located security patches using external reference URLs in CVE records. However, these approaches only address a limited subset of disclosed vulnerabilities due to often incomplete or incorrect vulnerability or code commit information [3]. Inspired by the semantic association between vulnerability description and code commits, learning-based methods employ deep learning and data mining techniques to discriminate the correlation between the vulnerability description and the code commit. For example, Zhou et al. [9] utilized word2vec [14] to learn the representations of the commit message and bug reports for security patch localization. Both Sabetta et al. [10] and Zhou et al. [11] employed the Long Short-Term Memory (LSTM) model [15] to separately encode the message and code change, aiding in the identification of security patches.

[*] Corresponding author

The emerging pre-trained language model (PLM) is trained on a very large-scale corpus, which can deeply understand the semantic information of the code, and has become the main method of encoding text information in the code [16]–[18]. For example, Nguyen et al. [13] employed BERT-based models [19] to encode commit messages, bug reports, and code changes, aiming to classify vulnerability-fixing commits. Similarly, Wang et al. [20] utilized CodeBERT [17] to extract semantic correlations between vulnerability descriptions and code commits. They enhanced their methods by integrating semantic correlation features with 36 handcrafted features to locate security patches.

Despite their promising results, learning-based localization methods still have two limitations: **(1) They cannot perform well in data scarcity scenarios.** Typically, existing learning-based approaches rely on the large-scale corpus to fine-tune the PLM. Nevertheless, the number of disclosed vulnerabilities with security patches remains limited. Numerous vulnerabilities are silently fixed in the open-source community and do not reveal corresponding patches [4]. The scarcity of security patches for particular types or programming languages of vulnerabilities amplifies the challenge. **(2) They are less effective in exploring semantic correlations between vulnerability descriptions and code commits.** Code commits contain two types of information: commit messages and code changes. Commit messages typically include file paths and descriptions of code changes, which are more domain-specific and noisy than other typical texts, such as social media posts or product reviews [21]. Code changes comprise a sequence of modified code statements extracted from one or multiple source code files. A difference exists between programming languages (in the form of code changes) and natural languages (including vulnerability descriptions and commit messages) [11]. Existing learning-based approaches employ a single model to handle both types of information. These approaches assume the same model can effectively capture the semantic correlations between vulnerability descriptions, commit messages, and code changes. However, given the different semantics of commit messages and code changes, this assumption may not always hold.

To alleviate the above two limitations, we propose a novel dual prompt tuning channel to locate the security patch called **PromVPat**, which is inspired by the recent advances in prompt tuning [22]. This approach captures semantic correlations between vulnerability descriptions and code commits, particularly in the data scarcity scenario. To tackle the first challenge, we utilize the prompt tuning method to fine-tune PLMs and guide PLMs' outputs in scarcity scenarios (i.e., few-shot learning). Prompt tuning adjusts the model's responses to prompts. Instead of training the model from scratch, prompt tuning uses a natural language instruction (i.e., "prompt") to hint at the downstream task. By modifying the original input and leveraging the pre-existing knowledge acquired by PLMs during the pre-training stage, prompt tuning can direct PLMs to focus on matching relevant information even with limited training data. To address the second limitation, we propose a novel dual prompt tuning channel with two prompts: code change and commit message channels. The former captures the semantic correlations between vulnerability descriptions and code changes. The latter focuses on the semantic correlations between vulnerability descriptions and commit messages. In addition to the abstract semantic features captured by the dual prompt tuning channel, we design 26 handcrafted features to capture the explicit correlation and adopt the attention mechanism to fuse the semantic and handcrafted features.

Specifically, given a vulnerability and a code commit, PromVPat first parses the code commit into the code change and the commit message. Then, PromVPat utilizes two prompt generators to generate the code change and commit message prompt to link them with vulnerability descriptions. The two prompts are initial prompts for prompt tuning of PLMs. Each prompt with the vulnerability and commit information is fed into an encoder (i.e., CodeT5 [16]) to capture the deep semantic correlations between the vulnerability descriptions and code commits. Considering the semantics between vulnerability descriptions and code commits, a fixed prompt (i.e., hard prompt) may lack flexibility or adaptability [23]. We adopt the soft prompt tuning method [24] in the dual prompt tuning channel, forming a continuous prompt with a learnable embedding. Soft prompt involves trainable parameters to the prompt itself, which are learned alongside the other parameters of the model during fine-tuning. Besides, we extract 26 handcrafted features that fall into two groups: vulnerability identifier and location features. These features explicitly capture the relationships between vulnerabilities and code commits. Further, we adopt an attention mechanism [25] to highlight the different importance of the handcrafted and semantic features and fuse them into the final correlation features. Finally, a multi-layer perceptron (MLP) is leveraged to produce the final correlation probability.

To evaluate the effectiveness of PromVPat, we compare it with five baselines across two public datasets, the VCMatch and SAP datasets, which serve as widely recognized benchmarks in locating security patches. Experiment results reveal that our approach outperforms all baselines by large margins regarding Recall and NDCG. Specifically, PromVPat improves the best-performing baseline by 14.42% and 86.57% across two datasets in Recall@1. Further analysis demonstrates the effectiveness of our approach in two data scarcity scenarios, namely, the cross-language and cross-project low-resource scenarios.

In summary, we make the following contributions:

- We propose a novel security patch localization approach named PromVPat, which takes the advantages of prompt engineering and few-shot learning [26] to locate security patches, especially in data scarcity scenarios.
- We conduct extensive experiments on two datasets to evaluate the superiority of the proposed model.
- We release our replication package [27], which includes the source code, the dataset, and our evaluation results.

II. Preliminaries

In this section, we first introduce the background knowledge related to vulnerabilities and prompt tuning. Then, we provide

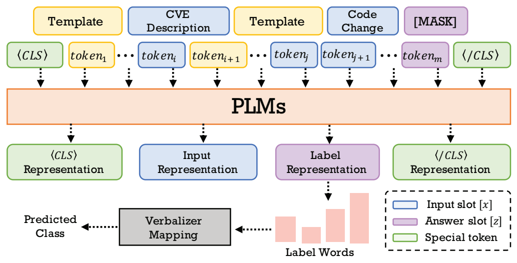


Fig. 1. An illustration of prompt tuning in the context of security patch localization. The blue rectangle symbolizes the input slot $[x]$ (e.g., vulnerability descriptions and code commits). The purple rectangle denotes the answer slot $[z]$ (e.g., "yes" and "no"). The green rectangle signifies the special token (e.g., $\langle CLS \rangle$).

a motivation example to show our key idea.

A. Background

1) Common Vulnerabilities and Exposure (CVE): CVE serves as a dictionary of identified vulnerabilities [28]. Security vendors can contact the CVE Assignment Team to request a unique CVE ID upon identifying a vulnerability. In addition to the CVE ID and vulnerability description, each CVE record includes a list of pertinent external references, such as security advisories and patches.

2) Prompt Tuning: Our approach builds upon the dual prompt tuning channel, which adopts the prompt tuning to stimulate PLMs' ability to capture semantic correlations between vulnerability descriptions and code commits. The process of prompt tuning is depicted in Figure 1. The underlying idea of prompt tuning is to align the training objective of downstream tasks with the pre-training phase of PLMs [22], [29]. The prompt tuning modifies the original input by introducing a natural language prompt to ensure the input format remains consistent with the pre-training stage.

Specifically, prompt tuning utilizes a prompt template, denoted as $f_{prompt}(x)$, to restructure the original input x and generate a new input x' with certain unfilled slots. As depicted in Figure 1, the prompt template comprises two types of reserved slots: input slot $[x]$ and answer slot $[z]$. The input slot $[x]$ is designated to be populated with the original input text, such as vulnerability descriptions and code changes. The answer slot $[z]$ is intended to be filled with predicted labels, such as "yes" and "no". We define the set of permissible values for input slots as Z, a small subset representing the classification labels. For example, in the security patch localization task, Z={"yes", "no", "$relevant$", "$irrelevant$"}. The PLM calculates the probability of the corresponding filled answer words and seeks the highest-scoring potential option $\hat{z} \in Z$, within the permissible values for Z.

$$\hat{z} = argmax(P(f_{prompt}(x); \theta)). \tag{1}$$

Inspired by previous works [22], [29], there are two main types of prompts: the hard prompt (a.k.a discrete prompt) and the soft prompt (a.k.a continuous prompt). The **hard prompt** [30]–[32] is the most natural method to create intuitive templates and modify the model input manually. The prompt is a text string where each token is meaningful. For instance, in the security patch localization task, the original input can be converted into "*The CVE* $[x_1]$. *The commit* $[x_2]$. *Relevant* $[z]$." $[x_1]$ denotes the vulnerability description, and $[x_2]$ represents the cod commit. The task objective becomes predicting the word at the answer slot $[z]$, such as "yes" or "true". The **soft prompt** [31], [33], [34] is different from the hard prompt in that the tokens of the prompt are represented as continuous vectors instead of fixed discrete words. We use $[soft]$ to denote the tokens in the soft prompt. The original hard prompt of the security patch localization task can be converted into

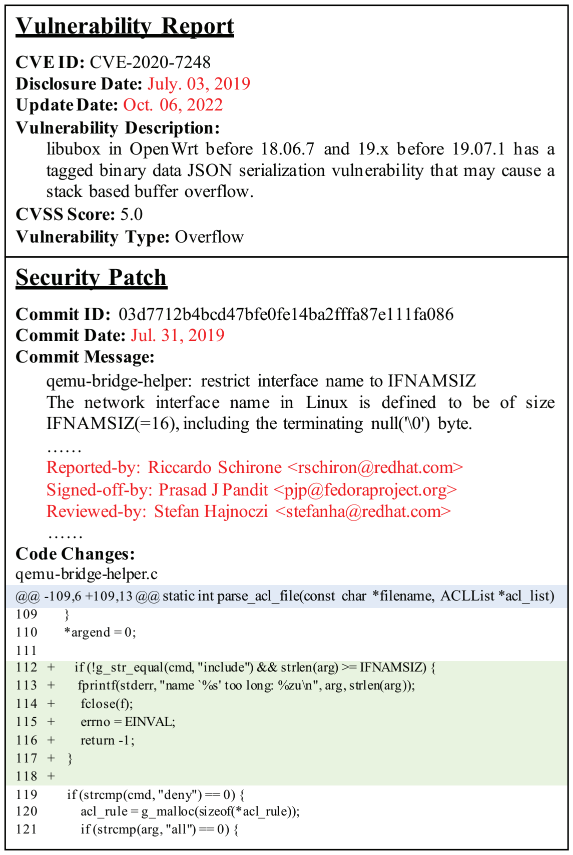
$$f_{prompt}(x_1, x_2) = [soft]\ [x_1][soft]\ [x_2]\ [soft]\ [z], \tag{2}$$

where $[soft]$ can be optimized during the tuning stage. Although hard prompt has shown promising performance in previous work [35], [36], it is time-consuming to create and discover optimal prompt templates [37], [38]. Since soft prompt is a continuous vector involving trainable parameters to the prompt representation, it allows for a wider range of prompts to help PLMs better adapt to the specific inputs. Besides, the soft prompt can capture more general patterns in the data. Hence, we adopt the soft prompt in our dual prompt tuning channel to stimulate the potentiality of PLMs.

B. Motivation Example

Figure 2 illustrates an example of a disclosed vulnerability and its corresponding security patch from the `qemu` project [39]. This vulnerability, identified as CVE-2020-7248 [40], could result in an over-read of data, subsequently causing a segmentation fault. Although its security patch was committed on July 31, 2019, it was not disclosed until October 06, 2022. **The disclosure of the patch occurred later than its submission.** This poses a risk to projects that depend on it, as the vulnerability provides potential attackers a substantial window of opportunity to launch attacks. It is crucial to locate the security patch of the disclosed vulnerability, enabling downstream projects to update their versions and implement remediation measures in time.

Further, we have the following findings: (1) The vulnerability description lacks essential identifier information of the commit, such as commit-ID, file names, and function names. Additionally, key vulnerability information such as CVE-ID and CWE-ID were not included in the code commit and could not be found through search-based localization methods. (2) The commit message is cluttered with extraneous information, including reporters' and reviewers' names and email details. This abundance of noise adversely affects the accuracy of matching vulnerability descriptions with code commits. (3) A substantial disparity exists between code changes and commit messages. Combining the two makes it challenging for existing models to capture the correlation between vulnerabilities and patches effectively. Given these challenges, we propose the dual prompt tuning channel to separately capture the semantic correlations between commit messages, code changes, and vulnerability descriptions.


Vulnerability Report

CVE ID: CVE-2020-7248
Disclosure Date: July. 03, 2019
Update Date: Oct. 06, 2022
Vulnerability Description:
libubox in OpenWrt before 18.06.7 and 19.x before 19.07.1 has a tagged binary data JSON serialization vulnerability that may cause a stack based buffer overflow.
CVSS Score: 5.0
Vulnerability Type: Overflow

Security Patch

Commit ID: 03d7712b4bcd47bfe0fe14ba2fffa87e111fa086
Commit Date: Jul. 31, 2019
Commit Message:
qemu-bridge-helper: restrict interface name to IFNAMSIZ
The network interface name in Linux is defined to be of size IFNAMSIZ(=16), including the terminating null('\0') byte.
……
Reported-by: Riccardo Schirone <rschiron@redhat.com>
Signed-off-by: Prasad J Pandit <pjp@fedoraproject.org>
Reviewed-by: Stefan Hajnoczi <stefanha@redhat.com>
……
Code Changes:
qemu-bridge-helper.c

```
@@ -109,6 +109,13 @@ static int parse_acl_file(const char *filename, ACLList *acl_list)
       }
       *argend = 0;

+       if (!g_str_equal(cmd, "include") && strlen(arg) >= IFNAMSIZ) {
+         fprintf(stderr, "name `%s' too long: %zu\n", arg, strlen(arg));
+           fclose(f);
+           errno = EINVAL;
+           return -1;
+       }
+
        if (strcmp(cmd, "deny") == 0) {
            acl_rule = g_malloc(sizeof(*acl_rule));
            if (strcmp(arg, "all") == 0) {
```

Fig. 2. An example of disclosed vulnerability and its security patch.

III. Approach

In this section, we introduce PromVPat, a novel security patch localization approach. We first introduce its overall architecture. Then, we describe the details of its three modules.

A. Overview

Figure 3 presents the framework of the PromVPat framework, composed of three modules: semantic matching, handcrafted feature extraction, and feature fusion & patch location.

Semantic Matching Module. This module utilizes a PLM and the dual prompt tuning channel to identify semantic correlations between vulnerability descriptions and code commits. It generates two new inputs using prompt templates and employs the PLM to produce two correlation probabilities, which serve as semantic features.

Handcrafted Feature Extraction Module. This module seeks to capture the explicit correlations between vulnerability descriptions and code commits. The handcrafted features draw on the prior work [20]. Notably, we extract fewer features, while the performance of PromVPat outperforms the earlier work [20], as presented in Section IV-C.

Feature Fusion & Patch Location Module. This module uses an attention mechanism to integrate the features derived from the previous two modules. This process emphasizes the varying significance of the semantic and handcrafted features. Finally, a classifier predicts the correlation probability to locate the security patch.

B. Semantic Matching Module

The dual prompt tuning channel, comprising the code change and commit message channels, identifies semantic correlations between vulnerability descriptions and commits. The code change channel focuses on the correlation between the code change and the vulnerability description, and the commit message channel targets the correlation between the commit message and the vulnerability description. Both channels contain three components: a prompt generator, encoder, and classifier.

Prompt Generator. This component transforms the original input into a new format using the prompt template. As discussed in Section II-A, we use the soft prompt, representing the prompt with continuous vectors. Following the insights from previous studies [41], [42], proper initialization of the $[soft]$ token can offer a beneficial starting point for optimizing the input embeddings. We use a manually designed prompt template to initialize $[soft]$ token representations, determining their number and position, rather than random initialization. As for the two channels, we design different initialization prompt templates. The template of the code change channel "*The CVE* $[x_1]$ *is fixed by the code* $[x_3]$. $[z]$" is converted into

$$f_{prompt}(x_1, x_3) = [soft]\ [x_1]\ [soft]\ [x_3]\ [z], \tag{3}$$

where $[x_1]$ represents the vulnerability description, and $[x_3]$ denotes the code change.

The prompt template of the commit message channel is "*The CVE:* $[x_1]$ *means* $[x_4]$. *Is it correct?* $[z]$". We convert it into

$$f_{prompt}(x_1, x_4) = [soft]\ [x_1]\ [soft]\ [x_4]\ [soft]\ [z], \tag{4}$$

where $[x_4]$ denotes the commit message. In our experiments, we compare four templates for each channel to select the optimal template initialization tokens.

Encoder. For each channel, we use CodeT5 [16] as the encoder to generate the input representations, including the input text (i.e., $[x_1]$, $[x_3]$, and $[x_4]$) and the prompt representation (i.e., $[soft]$). Initially, we freeze the CodeT5's parameters and derive the embeddings of the prompt tokens $\mathbf{P} \in \mathbb{R}^{p \times d}$, where p is the prompt length. For two input texts (e.g., $[x_1]$ and $[x_3]$), CodeT5 embeds their tokens into a matrix $\mathbf{X} \in \mathbb{R}^{2n \times d}$, where n is the token count of one input, and d is the size of embedding space dimension. The prompt embeddings are then appended to the input embeddings to create a single matrix $[\mathbf{P}; \mathbf{X}] \in \mathbb{R}^{(p+2n) \times d}$, fed into CodeT5 to maximize the answer words' probability. We evaluate three popular PLMs in our experiments: CodeBERT [17], CodeT5 [16], and GPT2 [43].

Classifier. A softmax classifier uses the learned input text representation to determine the answer word distribution. Then, the verbalizer M maps various searched answer words to a

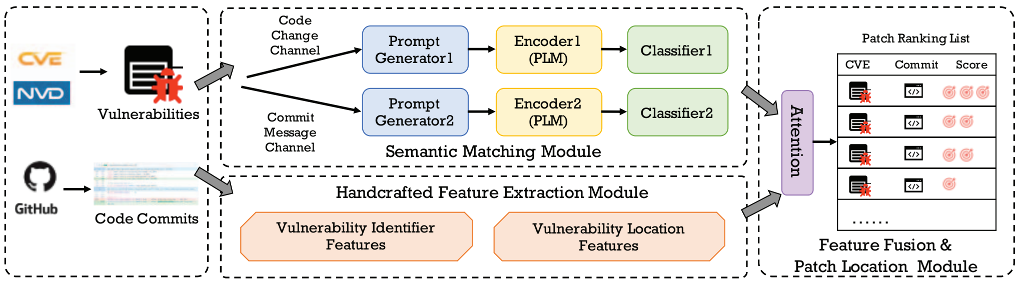


Fig. 3. An overview framework of the proposed model.

TABLE I
THE LIST OF HANDCRAFTED FEATURES.

Feature Type	Feature Name	Feature Description
Vulnerability Identifier Features	Number_of_CVE	Number of CVE IDs in commit messages
	Number_of_Bug	Number of bug IDs in commit messages
	Flag_of_CVE_Match	Whether the CVE IDs in commit messages
	Flag_of_Bug_Match	Whether Bug IDs in commit messages
	Flag_of_CWE_Match	Whether CWE IDs in commit messages
	Same_number_of_CVE_Msg	Number of same words between CVE description and commit messages
	Same_ratio_of_CVE_Msg	Ratio of same words between CVE description and commit message
	Max_of_same_CVE_Msg_Word	Max frequencies of same words between CVE descriptions and commit messages
	Sum_of_same_CVE_Msg_Word	Sum frequencies of same words between CVE descriptions and commit messages
	Average_of_same_CVE_Msg_Word	Average frequencies of same words between CVE descriptions and commit messages
	Variance_of_same_CVE_Msg_Word	Variance frequencies of same words between CVE descriptions and commit messages
	Same_number_of_CVE_Code	Number of same words between CVE descriptions and code changes
	Same_ratio_of_CVE_Code	Ratio of same words between CVE descriptions and code changes
	Max_of_same_CVE_Code_Word	Max frequencies of same words between CVE descriptions and code changes
	Sum_of_same_CVE_Code_Word	Sum frequencies of same words between CVE descriptions and code changes
	Average_of_same_CVE_Code_Word	Average frequencies of same words between CVE descriptions and code changes
	Variance_of_same_CVE_Code_Word	Variance frequencies of same words between CVE descriptions and code changes
Vulnerability Location Features	Number_same_file_path	Number of same file paths between CVE descriptions and commits
	Ratio_same_file_path	Ratio of same file paths between CVE descriptions and commits
	Number_unrelated_file_path	Number of file paths that is in commits but not mentioned in CVE descriptions
	Number_same_file	Number of same files between CVE descriptions and commits
	Ratio_same_file	Ratio of same files between CVE descriptions and commits
	Number_unrelated_file	Number of files in commits but not mentioned in CVE descriptions
	Number_same_function	Number of functions between CVE descriptions and commits
	Ratio_same_function	Ratio of functions between CVE descriptions and commits
	Number_unrelated_function	Number of functions in commits but not mentioned in CVE descriptions

single class. Specifically, our verbalizer maps the probability of answer words "true" or "yes" to the relevant class and "false" or "no" to the irrelevant class. We use the cross entropy loss to train the dual prompt tuning channel and predict correlation probabilities as deep semantic features.

C. Handcrafted Feature Extraction Module

In addition to semantic features, our approach also incorporates 26 handcrafted features to strengthen the correlation between the vulnerability description and the commit, as validated by previous studies [3], [20]. Table I provides details of these features, which fall into two categories: vulnerability identifier features and vulnerability location features. We further elaborate on these features below.

Vulnerability Identifier Features. Beyond CVE-ID and Bug-ID, we also consider the type and impact of the vulnerability as identifiers, such as CWE-ID, the CWE description, vulnerability trigger conditions, and vulnerability causes. We employ corresponding regular expressions (i.e., "CVE-[0-9]4-[0-9]{1, 8}" and "bug.0,3([0-9]{2, 5}") to discern if the commit is related to the vulnerability identifier and count the number of these identifiers in the commit message. Besides, we tokenize the commit message, the code change, and the vulnerability description. We then count the identical tokens between the commit message and the vulnerability description, and between the code change and the vulnerability description. Further, we compute five statistical features (i.e., ratio, maximum, sum, average, and variance). Finally, we identify 17 features between the vulnerability and the commit.
Vulnerability Location Features. Six features are derived from the number and ratio of the related files and functions between the vulnerability description and the commit. We count the number of file paths, files, and functions that do not relate to the vulnerability description. Vulnerability location features enhance the correlation by verifying if the code file or function, where the vulnerability is situated, is present in the code commit.

D. Feature Fusion & Patch Location Module

To indicate the varying importance of different features, we apply the dot-production attention mechanism [25] to merge the extracted features and produce the attentive correlation features $\hat{C}$. Finally, we use an MLP classifier to generate the final correlation probability $\hat{y}$, as follows:

$$\hat{y} = \mathrm{Softmax}(\mathrm{MLP}(\tanh(\hat{C}))). \tag{5}$$

E. Model Training

We use the cross entropy loss to forecast the correlation and train our approach by minimizing the following loss:

$$\mathcal{L}(\hat{y}, y) = -y \cdot log\left(\hat{y}\right) + (1-y) \cdot log\left(1-\hat{y}\right), \tag{6}$$

where y represents the actual value. If a connection exists between the vulnerability description and the security patch, y is set to 1; otherwise, it is 0.

IV. Experiment

In this section, we conduct extensive experiments to justify our model's superiority and analyze the reasons for its effectiveness. We aim to answer the following research questions:
RQ1: How effective is PromVPat compared to the state-of-the-art baselines on locating security patches for disclosed vulnerabilities?
RQ2: What are the effects of different prompt tuning design choices for the proposed model?
RQ3: What are the effects of different handcrafted features for the proposed model?
RQ4: Can PromVPat outperform existing localization approaches in data scarcity scenarios?

A. Dataset

To evaluate the effectiveness of PromVPat, we adopt two public datasets: the VCMatch [20] and SAP dataset [44]. The VCMatch and SAP datasets serve as widely recognized benchmarks in security patch localization. The VCMatch dataset proposed by Wang et al. [20] comprises 1,318 vulnerabilities from 10 large-scale open-source projects. The SAP dataset proposed by Ponta et al. [44] includes 566 vulnerabilities covering 205 distinct Java open-source projects. Each dataset is randomly divided into three segments: 80% for training, 10% for validation, and the final 10% for testing. Table II provides a comprehensive statistical overview of these two datasets.

TABLE II
THE STATISTICS OF DATASETS.

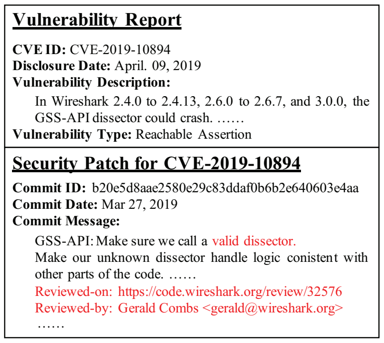
Dataset	VCMatch	SAP
# Vulnerability	1,318	566
# Total Commits	705,456	7,165
# Training Commits	564,364	5,719
# Validation Commits	70,546	756
# Test Commits	70,546	690

B. Experimental Setup

Baselines. We compare PromVPat with five state-of-the-art approaches: XGBoost [45], LightGBM [46], PATCHSCOUT [3], VCMatch [20], and VulCurator [13]. These approaches can be categorized into three types: traditional classification models, neural-network-based models, and PLMs-based models. **XGBoost** and **LightGBM** fall under the category of traditional classification models. They are both tree-based learning algorithms that belong to the gradient-boosting framework. We employ these two models to predict the correlation between patches and vulnerability descriptions based on 22 manually extracted features. **PATCHSCOUT** is a neural-network-based localization model that utilizes RankNet [47] and 22 handcrafted features to locate security patches. **VCMatch** is a PLMs-based model that employs CodeBERT to capture semantic features from vulnerability descriptions and commit messages. Besides, it considers 36 handcrafted features to locate security patches. **VulCurator** is also a PLMs-based model that utilizes three CodeBERT models to predict the probabilities of code messages, code changes, and bug reports related to vulnerabilities. We do not compare PromVPat with matching-based methods since previous studies [3], [20] have demonstrated that matching-based methods have limited effectiveness.

Evaluation Metrics. Our model's primary goal is ranking items based on relevance rather than classifying them into distinct categories. In ranking models, the emphasis is on the relative order of items rather than their discrete categorization. Using classification-based metrics like precision, accuracy, and F1 would not capture the nuances and intricacies of ranking performance. We followed previous works [3], [20], adopted two widely-used Top-K ranking metrics instead of classification-based metrics, i.e., $Recall@K$ and Normalized Discounted Cumulative Gain ($NDCG@K$), for evaluation (K is 1 and 5). $NDCG = \frac{DCG}{IDCG}$, where $DCG = rel_1 + \sum_{i=2}^{n} \frac{rel_i}{log_2 i}$ and $IDCG = rel_{sorted_1} + \sum_{i=2}^{n} \frac{rel_{sorted_i}}{log_2 i}$. rel_i is the relevance of the item at position i. rel_{sorted_i} is the relevance of items in the ideal order. $Recall@K$ is the percentage of successfully identified security patches, while $NDCG@K$ factors in the positional success of the code commit, allocating a higher score if the security patch resides within the top positions. In all metrics, greater values signify superior performance. Furthermore, we present the mean score of all vulnerabilities within the test set as the conclusive score.

Implementation Details. We adopt the PyTorch framework [48], the Huggingface [49], and the OpenPrompt [50] to implement PromVPat. In the prompt tuning stage, we assign a value of 20 to the number of $[soft]$ tokens. The learning rate is 1e-5. The maximum length of each input is 512. Moreover, we choose AdamW [51] as the optimizer. During the inference process, we predict the correlation probability of each CVE and code commit and rank the order of code commits according to the probability. Besides, we maintain consistent parameter configurations when implementing the other baselines. The experiments are performed on a machine equipped with two


Vulnerability Report

CVE ID: CVE-2019-10894
Disclosure Date: April. 09, 2019
Vulnerability Description:
In Wireshark 2.4.0 to 2.4.13, 2.6.0 to 2.6.7, and 3.0.0, the GSS-API dissector could crash. ……
Vulnerability Type: Reachable Assertion

Security Patch for CVE-2019-10894

Commit ID: b20e5d8aae2580e29c83ddaf0b6b2e640603e4aa
Commit Date: Mar 27, 2019
Commit Message:
GSS-API: Make sure we call a valid dissector.
Make our unknown dissector handle logic conistent with other parts of the code. ……
Reviewed-on: https://code.wireshark.org/review/32576
Reviewed-by: Gerald Combs <gerald@wireshark.org>
……

Fig. 4. An example of disclosed vulnerability (CVE-2019-10894) and its security patch.

Nvidia GeForce GTX 3090 GPUs and two Intel Xeon Gold 6226R CPUs.

C. Effectiveness on Patch Localization (RQ1)

We compare our approach with other baselines on two datasets regarding Recall and NDCG. Table III illustrates the overall performance results. The results show that PromVPat outperforms all the baselines across two datasets in terms of four metrics by large margins, indicating the effectiveness of PromVPat. Specifically, it improves over the best-performing baseline, i.e., VulCurator, by 14.42% and 86.57% in two datasets in terms of $Recall@1$, respectively. PromVPat improves VulCurator on two datasets of $NDCG@1$ and $NDCG@5$ by 0.97%, 11.36%, 27.32%, and 2.23%, respectively. Compared with Recall, our model shows a small improvement in the results on NDCG. The reason is that the dataset has few relevant patches for each vulnerability. Further, we have the following observations:

(1) Traditional classifiers (i.e., XGBoost and LightGBM) are limited in locating the security patches. This could be attributed to their reliance on manually extracted features for localization, which cannot comprehend the underlying semantic information. In contrast, our approach not only adopts handcrafted features but also utilizes the PLMs to capture the deep semantic features between vulnerability description and code commits, which enhances the precision of security patch localization.

(2) It is evident that PLMs-based models (i.e., VCMatch, VulCurator, and PromVPat) outperform the other baseline models. This indicates that PLMs can effectively capture the semantic correlation between vulnerability descriptions and code commits. PromVPat utilizes the dual prompt tuning channel to capture the semantic correlations of the commit message and code changes with the vulnerability description. This allows PLMs to focus on the specific semantic features crucial to the matching process. To further explore why PromVPat can achieve better performance, we manually inspect the ranking results of PromVPat and the best-performing baseline, VulCurator. Our manual inspection shows PromVPat can identify key information related to vulnerabilities from noisy commit messages, improving the matching precision between vulnerabilities and commits. Figure 4 presents an example from our test set: a reachable-assertion-type vulnerability identified as CVE-2019-10894. The patch message for this vulnerability includes the reporter and reviewer's names and email addresses. Despite this additional information, our approach can still identify words related to the vulnerability, such as "valid dissector". This example further confirms the effectiveness of our approach in accurately localizing security patches.

D. Effects of Different Prompt Tuning Designs (RQ2)

In this subsection, we aim to explore the impact of different prompt-tuning design choices on the effectiveness of our approach, including different numbers of prompt-tuning channels, different prompt-tuning methods, different prompt templates, different answer words, and different PLMs.

1) The Effect of Dual Prompt Tuning Channel: The key idea of PromVPat is to adopt the dual prompt tuning channel and stimulate PLMs' ability to capture deep semantic features between vulnerability descriptions and code commits. To explore the effect of the dual prompt tuning channel, we compare PromVPat with its three variants: (1) **PromVPat-mess** only considers the prompt tuning in the message channel. (2) **PromVPat-code** only adopts the prompt tuning method in the code change channel. (3) **PromVPat-single** does not use dual prompt channels but directly stitches the vulnerability description and code commit together to calculate the association probabilities.

Table IV presents the results. PromVPat outperforms all its variants across four metrics. Specifically, for $Recall@1$, the improvements achieved by PromVPat range from 2.58% to 16.67%. These results suggest that the dual prompt tuning channel positively impacts the extraction of deep semantic features. Moreover, the performance improvement of PromVPat over PromVPat-mess in Recall is less than the improvement over PromVPat-code. One possible reason is that capturing the semantics between vulnerability descriptions and code changes is more difficult. Besides, the performance of PromVPat-single is worse than the other three models, indicating the importance of the dual prompt tuning channel.

2) The Effect of Different Prompt Tuning Methods: As we introduced in section II-A, there are two types of prompt tuning methods: hard and soft prompt. To study the impact of different prompt tuning methods, we compare PromVPat with three variants: (1) **PromVPat-hh** adopts the hard prompt tuning method in both channels. (2) **PromVPat-hs** utilizes the hard prompt tuning method in the code change channel and the soft prompt tuning method in the commit message channel. (3) **PromVPat-sh** employs the soft prompt tuning method in the code change channel and the hard prompt tuning method in the other channel. We conduct experiments with fixed answer words and PLMs.

The results of our comparisons are presented in Table V. We find that the prompt tuning methods would impact the model performance. Compared with hard prompt tuning, soft prompt tuning performs better. Our approach can locate the most patches when both channels use the soft prompt tuning.

TABLE III
PERFORMANCE COMPARISONS OF OUR APPROACH WITH OTHER BASELINES.

Model\Dataset	VCMatch Dataset				SAP Dataset			
	Recall@1	NDCG@1	Recall@5	NDCG@5	Recall@1	NDCG@1	Recall@5	NDCG@5
XGBoost	74.24%	31.72%	78.03%	32.74%	17.52%	13.33%	39.43%	30.14%
LightGBM	76.51%	32.69%	78.79%	33.29%	19.30%	14.22%	40.13%	33.35%
PATCHSCOUT	75.76%	32.36%	78.79%	33.18%	19.12%	21.57%	41.99%	34.17%
VCMatch	75.76%	32.36%	81.81%	34.95%	19.71%	22.21%	43.62%	35.73%
VulCurator	78.79%	37.54%	79.55%	26.52%	21.37%	27.45%	56.21%	44.92%
PromVPat	**90.15%**	**38.51%**	**91.67%**	**38.92%**	**39.87%**	**34.95%**	**66.01%**	**45.92%**
Improvement	14.42%	0.97%	12.05%	11.36%	86.57%	27.32%	17.43%	2.23%

TABLE IV
EVALUATION RESULTS OF THE EFFECT OF DUAL PROMPT TUNING CHANNEL.

Method	Metrics			
	R@1	N@1	R@5	N@5
PromVPat-mess	87.88%	37.54%	88.64%	37.74%
PromVPat-code	82.58%	35.28%	83.33%	35.40%
PromVPat-single	77.27%	33.01%	81.06%	34.03%
PromVPat	**90.15%**	**38.51%**	**91.67%**	**38.92%**

TABLE V
EVALUATION RESULTS OF DIFFERENT PROMPT TUNING METHODS.

Tuning Method	Metrics			
	Recall@1	NDCG@1	Recall@5	NDCG@5
PromVPat-hh	85.61%	36.57%	87.12%	36.90%
PromVPat-hs	86.36%	36.89%	89.39%	37.71%
PromVPat-sh	87.88%	37.54%	89.39%	37.95%
PromVPat	**90.15%**	**38.51%**	**91.67%**	**38.92%**

This may be because hard prompt tuning falls short in providing valuable guidance for PLMs. Hence, we choose both channels' soft prompt tuning methods to stimulate the PLMs' ability.

3) The Effect of Different Prompt Templates: Inspired by previous works [24], [42], the initial words of the prompt template are crucial for soft prompt tuning. We design four templates for each channel to investigate the effect of different prompt template words, shown in Table VI.

TABLE VI
EVALUATION RESULTS OF DIFFERENT PROMPT TEMPLATES.

Channel Type	Prompt Templates	Metrics	
		R@1	N@1
Code Change Channel	$[x_1]$ means $[x_3]$? Is it correct? [z].	88.64%	37.86%
	Code: $[x_3]$ fix $[x_1]$? Is it correct [z].	84.85%	36.24%
	Code: $[x_3]$ CVE: $[x_1]$ Relevant [z]	87.12%	37.22%
	CVE $[x_1]$ is fixed by code $[x_3]$ [z]	**90.15%**	**38.51%**
Commit Message Channel	$[x_1]$ means $[x_4]$? Is it correct? [z].	**90.15%**	**38.51%**
	Message $[x_4]$ describe $[x_1]$ Is it correct? [z]	84.85%	36.25%
	CVE: $[x_1]$ Message: $[x_4]$ Relevant [z]	84.09%	35.92%
	CVE $[x_1]$ is described by message $[x_4]$ [z]	86.36%	36.89%

The experimental results are shown in Table VI. We can observe that the choice of prompt templates significantly impacts our approach's effectiveness. Specifically, when we select the template "The CVE $[x_1]$ is fixed by the code $[x_3]$ $[z]$." and "$[x_1]$ means $[x_4]$? Is it correct? $[z]$." as the prompt for the code change and commit message channel, PromVPat achieves the best performance in all metrics. It indicates that each channel should focus on different semantic aspects of the code change and commit message, highlighting the deep semantic correlation between input texts. Consequently, it is necessary to design different prompt templates for two channels. The choice of prompt words plays a pivotal role in steering the response of language models. While two words may be synonymous, the contexts in which they are typically used can vary. We can enhance interactions' robustness, depth, and breadth with language models by employing different but synonymous prompt words, ensuring more accurate and contextually relevant outputs.

4) The Effect of Different Answer Words: In addition to the prompt template words, different answer words may also impact the performance of PromVPat. To explore the effect of different answer words, we maintain a consistent prompt tuning method, adopt the same template words, and select task-relevant answer words for the verbalizers.

TABLE VII
EVALUATION RESULTS OF DIFFERENT PROMPT VERBALIZERS.

Channel Type	Prompt Verbalizer	Metrics	
		Recall@1	NDCG@1
Code Change Channel	+: "yes", -: "no"	89.39%	38.19%
	+: "true", -: "false"	**90.15%**	**38.51%**
	+: "relevant", -: "irrelevant"	88.64%	37.86%
Commit Message Channel	+: "yes", -: "no"	**90.15%**	**38.51%**
	+: "true", -: "false"	87,88%	37.54%
	+: "relevant", -: "irrelevant"	88.64%	37.86%

As demonstrated in Table VII, when we utilize the answer words "*true*" and "*false*" to fill the answer slot $[z]$ of the code change channel and the answer words "*yes*" and "*no*" in the commit message channel, the metrics are superior to those obtained using other answer words. These results validate that selecting appropriate answer words can help capture the deep semantic correlations between the vulnerability description and code commits.

5) The Effect of Different PLMs: In PromVPat, we adopt the CodeT5 [16] to encode the input text. To explore the adaptability and performance of our approach with other PLMs, we integrated PromVPat with two other prominent PLMs: CodeBERT [17] and GPT2 [43]. To this end, we design two variants, PromVPat-CodeBERT and PromVPat-GPT2. These two models use CodeBERT and GPT2 to encode the input text. Our empirical evaluations, presented in Table VIII, reveal that when applied to PromVPat, PromVPat achieves superior results, outperforming its counterparts using PromVPat-CodeBERT and

PromVPat-GPT2 by an impressive margin of over 1.51 points in $Recall@1$. The reasons can be attributed to several key factors. (1) CodeT5 is trained on a larger and more diverse dataset than CodeBERT, which is more advantageous than CodeBERT. (2) GPT2 adopts unsupervised learning, which has not been specifically optimized for classification tasks, and may be less effective. (3) There is a fundamental architectural difference between GPT2 and CodeT5. GPT2 adopts the only-decoder architecture, while CodeT5 adopts the encoder-decoder architecture, which performs better in generating representations of $[soft]$ tokens and contributes to its superior performance in capturing deep semantic correlations.

TABLE VIII
EVALUATION RESULTS OF DIFFERENT PLMS.

PLM Type	Metrics	
	Recall@1	NDCG@1
PromVPat-CodeBERT	88.64%	37.86%
PromVPat	**90.15%**	**38.51%**
PromVPat-GPT2	84.09%	35.92%

E. The Effect of Handcrafted Features (RQ3)

As shown in Table I, our approach extracts 26 handcrafted features categorized into two groups. To measure the contribution of each group of handcrafted features, we compare PromVPat with two variants that only consider one type of handcrafted features.

TABLE IX
EVALUATION RESULTS OF DIFFERENT HANDCRAFTED FEATURES.

Feature Types	Metrics			
	R@1	N@1	R@5	N@5
Without Identifier	81.81%	34.95 %	84.09%	35.52%
Without Location	89.39%	38.19 %	90.15%	38.31%
PromVPat	**90.15%**	**38.51%**	**91.67%**	**38.92%**

Table IX demonstrates the effect of different categories of handcrafted features. Each category significantly boosts the performance of our approach. The vulnerability identifier features contribute the most. This is likely because this type of feature is unique to a vulnerability and can most directly establish the explicit relationship between a vulnerability and its security patches.

F. Effectiveness in Data Scarcity Scenarios (RQ4)

To study how well prompt tuning can handle data scarcity scenarios, we establish two data scarcity scenarios: (1) **cross-language low-resource scenario**, where we train on the C/C++ programming language and evaluate on PHP language. (2) **cross-project low-resource scenario**, where the model is trained on a limited number of datasets from various projects and then tested on the target projects. Following previous research [29], we first simulate a low-resource setting by randomly selecting 1, 4, 8, and 16 vulnerabilities per project with C/C++ or the project to create four training datasets (also referred to as 1, 4, 8, and 16 shots). We then compare the performance of prompt tuning with the fine-tuning method in these two scenarios.

1) Performance in the Cross-Language Low-Resource Scenario: Table X presents the $Recall@1$ achieved by both tuning methods across five training datasets. Our approach performs better than the model using fine-tuning in low-resource scenarios (i.e., 1, 4, 8, and 16 shots). On average, prompt tuning exceeds fine-tuning by 4.98%, 7.41%, 10.00%, and 4.65% across the four datasets in terms of $Recall@1$, respectively. These results confirm that prompt tuning can extract the knowledge of PLMs with minimal dataset training, as it transforms the task into a language model format.

TABLE X
EVALUATION RESULTS IN THE CROSS-LANGUAGE LOW-RESOURCE SCENARIO.

Method	Different Shots			
	1 Shot	4 Shots	8 Shots	16 Shots
Fine-Tuning	48.78%	49.39%	48.78%	52.44%
Prompt-Tuning	**51.21%**	**53.05%**	**53.66%**	**54.88%**

2) Performance in the Cross-Project Low-Resource Scenario: In certain projects, the amount of available training data is often limited. For example, in the VCMatch dataset, the OpenSSL project [52] only contains 92 vulnerabilities with publicly available security patches. To evaluate the performance of our approach under the cross-project low-resource setting, we train on 1-shot, 4-shot, 8-shot, and 16-shot settings, then evaluate the test set used in IV-C. Table XI shows the $Recall@1$ achieved by both PromVPat and the fine-tuning method. Prompt tuning achieves better performance than fine-tuning in all few-shot settings. Specifically, prompt tuning outperforms the fine-tuning method by 0.97%, 0.97%, 1.02%, and 8.73% across the four few-shot settings. Besides, we also observe that the performance improvement does not increase but rather decreases as the number of training shots increases. These results demonstrate that prompt tuning is more advantageous than fine-tuning when dealing with limited training data.

TABLE XI
EVALUATION RESULTS IN THE CROSS-PROJECT LOW-RESOURCE SCENARIO.

Method	Different Shots			
	1 Shot	4 Shots	8 Shots	16 Shots
Fine-Tuning	78.03%	78.03%	77.27%	78.03%
Prompt-Tuning	**78.79%**	**78.79%**	**78.03%**	**84.84%**

V. DISCUSSION

A. Handcrafted Features

VCMatch [20] and PatchScout [3] employ four feature groups: vulnerability identifier, vulnerability location, vulnerability type, and vulnerability descriptive texts to locate security patches. As shown in Table III, our approach outperforms VCMatch and PatchScout. To evaluate the impact of the four groups of handcrafted features, we design one variant named **PromVPat-af** that incorporates four types of vulnerability features and compare it with PromVPat. As indicated in Table XII, PromVPat performs better. We believe this is because the other two characteristics may lead to model overfitting. Our dual prompt tuning channel can more precisely capture the semantic correlation between vulnerability descriptions and code commits.

TABLE XII
EVALUATION RESULTS OF DIFFERENT HANDCRAFTED FEATURES.

Approach	Metrics			
	Recall@1	NDCG@1	Recall@5	NDCG@5
PromVPat-af	87.88%	37.54%	90.91%	38.27%
PromVPat	**90.15%**	**38.51%**	**91.67%**	**38.92%**

B. Other Auxiliary Information

Besides vulnerability descriptions and code commits, some studies incorporate auxiliary information, such as issue report contents, to strengthen the correlations between vulnerabilities and their patches [12], [13]. However, our datasets do not include issue report information, so we did not factor in issue reports when reproducing VulCurator. We investigated to determine the number of existing CVE records that contain issue report links by searching for related keywords like "issue" and "bug". Following the methodology of a previous study [53], we collected CVE records from 2002 to 2022 from the CVEdetail website [54]. Our findings reveal that only **19.61%** of CVE records contain issue report links. This result represents the maximum possible coverage, as this method may include incorrect or incomplete links. This suggests that auxiliary information can only encompass a small portion of CVE records.

C. Time Efficiency

Table XIII outlines the time costs of PromVPat on the VCMatch dataset during two stages: model training and patch localization. It indicates that PromVPat necessitates approximately 3.1 hours for one epoch of training on the VCMatch dataset. Given that the training is typically conducted offline, we think that the time required is reasonable. On average, PromVPat takes less than 9.77 seconds to identify one patch from 1,000 code commits, suggesting that PromVPat's efficiency is adequate for real-world applications.

TABLE XIII
TIME COSTS OF PROMVPAT ON VCMATCH DATASET.

Approach	Training	Location
PromVPat	3.1 hours/epoch	9.77 seconds/1k

D. Threats to Validity

1) Internal Validity: Internal validity threats pertain to potential issues with our study's execution. One such threat is the potential bias in the experiment results due to the limited number of datasets used. To address this, we conducted experiments using two datasets spanning three programming languages: C/C++, PHP, and Java. We plan to incorporate more datasets in future evaluations of our approach. Another validity threat is our inability to perform extensive hyper-parameter optimizations on our model due to hardware constraints, a common issue in deep learning model work. We have mitigated this threat by following the hyper-parameter values used in previous work [29]. Furthermore, we have ensured that the experimental setups of baselines are reasonable and align closely with their descriptions in the replication package.

2) External Validity: External validity threats relate to potential issues with the generalizability of our approach. One such threat is that we adopt CodeT5 [16] as the encoder to learn input representations, instead of other PLMs. However, existing experiments and results (refer to Section IV-D5) have validated the superiority and universality of our approach. Additionally, our approach uses manually designed template words as initialization to represent the $[soft]$ token in the prompt tuning process. We mitigate this threat by creating four templates for each channel and comparing them. The experimental results (refer to Section IV-D3) demonstrate that our approach excels in locating security patches. Future research should concentrate on automatically selecting suitable initial words from a large corpus to enhance the generalizability of our approach.

VI. Related Work

A. Security Patches Localization

Numerous studies have explored the localization of security patches [3], [8], [55], [56]. Early research primarily concentrated on identifying security patches [8], [55]. Xu et al. [56] employed the regular expression to identify the security patch. Wang et al. [8] mined additional code change features and assembled five traditional classification algorithms for patch identification. Several studies also attempted to leverage the correlations between vulnerability descriptions and code commits [3]. For example, PATCHSCOUT [3] used RankNet [47] to locate the security patch based on various handcrafted features between vulnerability descriptions and code commits. As PLMs in NLP have evolved, researchers have attempted to use PLMs to identify security patches and establish the correlation between the vulnerability description and the commit [4], [13], [20]. For instance, Zhou et al. [4] employed CodeBERT [17] to extract semantic representation from code changes and identify silent security patches. Nguyen et al. [13] introduced a security patch location tool that uses the BERT-based model [17] to encode the commit message, code changes, and issue reports, and generates the probability that the commit is the security patch. Wang et al. [20] used CodeBERT [17] to capture the semantic correlations of the vulnerability descriptions and commit messages, and combined the probabilities of three traditional classifiers to locate the security patch. Unlike these models, our approach employs the prompt tuning method to maximize the potential of existing PLMs in capturing deep semantic features between vulnerabilities and commits. Moreover, our proposed approach can deliver superior performance in scenarios with scarce data.

B. Prompt Tuning in Software Engineering

Prompt tuning has gained significant popularity in recent years, driven by the advent of large PLMs. The software engineering community has conducted extensive explorations of prompt tuning [26], [29], [57]–[61]. For example, Wang et al. [29] evaluated prompt tuning experimentally on CodeBERT [17] and CodeT5 [16] in three code intelligence tasks: defect prediction, code summarization, and code translation.

The results demonstrated that prompt tuning surpasses fine-tuning in these tasks. Le et al. [57] employed prompt tuning to identify keywords and parameters in log messages. Huang et al. [58] created a dynamic prompt generator to extract API entities and relations using a PLM. Liu et al. [60] developed a prompt-based input generation and tuning method to automatically generate GUI testing text based on the GUI page's text and view hierarchy file. In our work, we introduce a dual prompt tuning approach to enhance the generalization ability of text and code matching models and improve the performance of locating security patches for disclosed vulnerabilities, particularly in situations with limited training data. To our knowledge, this is the first initiative in this direction.

VII. Conclusion and Future Work

This paper introduces PromVPat, a novel approach to localizing security patches for disclosed OSS vulnerabilities. The key idea of PromVPat involves using a dual prompt tuning channel to extract semantic features between vulnerability descriptions and code commits. Besides, PromVPat employs 26 handcrafted features and an attention mechanism to fuse the semantic and handcrafted features, capturing their correlation. Evaluations of two datasets demonstrate that PromVPat outperforms all existing methods, particularly in situations with limited data. In future work, we plan to extend the application of the dual prompt tuning channel to other tasks that require two inputs or are faced with data scarcity scenarios.

ACKNOWLEDGMENT

This research is supported by the Fundamental Research Funds for the Central Universities (No. 226-2022-00064), the National Natural Science Foundation of China (No. 62141222), and the National Natural Science Foundation of China (No. 62372398).

References

[1] whitesource, "Open source vulnerability management report," in *https://www.whitesourcesoftware.com/open-source-vulnerability-management-report/*, 2020.

[2] X. Cheng, H. Wang, J. Hua, G. Xu, and Y. Sui, "Deepwukong: Statically detecting software vulnerabilities using deep graph neural network," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 3, pp. 1–33, 2021.

[3] X. Tan, Y. Zhang, C. Mi, J. Cao, K. Sun, Y. Lin, and M. Yang, "Locating the security patches for disclosed oss vulnerabilities with vulnerability-commit correlation ranking," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3282–3299.

[4] J. Zhou, M. Pacheco, Z. Wan, X. Xia, D. Lo, Y. Wang, and A. E. Hassan, "Finding A needle in a haystack: Automated mining of silent vulnerability fixes," in *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 2021, pp. 705–716.

[5] K. Vaniea and Y. Rashidi, "Tales of software updates: The process of updating software," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, May 7-12, 2016*. ACM, 2016, pp. 3215–3226.

[6] 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 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 595–614.

[7] F. Li and V. Paxson, "A large-scale empirical study of security patches," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. ACM, 2017, pp. 2201–2215.

[8] X. Wang, K. Sun, A. L. Batcheller, and S. Jajodia, "Detecting "0-day" vulnerability: An empirical study of secret security patch in OSS," in *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Portland, OR, USA, June 24-27, 2019*. IEEE, 2019, pp. 485–492.

[9] Y. Zhou and A. Sharma, "Automated identification of security issues from commit messages and bug reports," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*. ACM, 2017, pp. 914–919.

[10] A. Sabetta and M. Bezzi, "A practical approach to the automatic classification of security-relevant commits," in *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018*. IEEE Computer Society, 2018, pp. 579–582.

[11] Y. Zhou, J. K. Siow, C. Wang, S. Liu, and Y. Liu, "SPI: automated identification of security patches via commits," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 1, pp. 13:1–13:27, 2022.

[12] G. Nguyen-Truong, H. J. Kang, D. Lo, A. Sharma, A. E. Santosa, A. Sharma, and M. Y. Ang, "HERMES: using commit-issue linking to detect vulnerability-fixing commits," in *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022, Honolulu, HI, USA, March 15-18, 2022*. IEEE, 2022, pp. 51–62.

[13] T. G. Nguyen, T. Le-Cong, H. J. Kang, X. D. Le, and D. Lo, "Vulcurator: a vulnerability-fixing commit detector," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*. ACM, 2022, pp. 1726–1730.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[15] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, 1997.

[16] Y. Wang, W. Wang, S. R. Joty, and S. C. H. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*. Association for Computational Linguistics, 2021, pp. 8696–8708.

[17] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "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*, ser. Findings of ACL, vol. EMNLP 2020. Association for Computational Linguistics, 2020, pp. 1536–1547.

[18] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019.

[19] J. Devlin, M. 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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 2019, pp. 4171–4186.

[20] S. Wang, Y. Zhang, L. Bao, X. Xia, and M. Wu, "Vcmatch: A ranking-based approach for automatic security patches localization for oss vulnerabilities," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 589–600.

[21] J. Wang, Z. Wang, D. Zhang, and J. Yan, "Combining knowledge with deep convolutional neural networks for short text classification," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. ijcai.org, 2017, pp. 2915–2921.

[22] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Comput. Surv.*, vol. 55, no. 9, pp. 195:1–195:35, 2023.

[23] B. Lester, R. Al-Rfou, and N. Constant, "The power of scale for parameter-efficient prompt tuning," 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*. Association for Computational Linguistics, 2021, pp. 3045–3059.

[24] G. Qin and J. Eisner, "Learning how to ask: Querying lms with mixtures of soft prompts," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*. Association for Computational Linguistics, 2021, pp. 5203–5212.

[25] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*. The Association for Computational Linguistics, 2015, pp. 1412–1421.

[26] N. Nashid, M. Sintaha, and A. Mesbah, "Retrieval-based prompt selection for code-related few-shot learning," in *Proceedings of the 45th International Conference on Software Engineering (ICSE'23)*, 2023.

[27] "Replication package," https://doi.org/10.5281/zenodo.8202620, 2023.

[28] CVE, "Common vulnerabilities and exposure," in *https://cve.mitre.org/*, 2023.

[29] C. Wang, Y. Yang, C. Gao, Y. Peng, H. Zhang, and M. R. Lyu, "No more fine-tuning? an experimental evaluation of prompt tuning in code intelligence," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*. ACM, 2022, pp. 382–394.

[30] Y. Gu, X. Han, Z. Liu, and M. Huang, "PPT: pre-trained prompt tuning for few-shot learning," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*. Association for Computational Linguistics, 2022, pp. 8410–8423.

[31] X. Han, W. Zhao, N. Ding, Z. Liu, and M. Sun, "PTR: prompt tuning with rules for text classification," *AI Open*, vol. 3, pp. 182–192, 2022.

[32] T. Schick and H. Schütze, "Exploiting cloze-questions for few-shot text classification and natural language inference," in *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*. Association for Computational Linguistics, 2021, pp. 255–269.

[33] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*. Association for Computational Linguistics, 2021, pp. 4582–4597.

[34] M. Tsimpoukelli, J. Menick, S. Cabi, S. M. A. Eslami, O. Vinyals, and F. Hill, "Multimodal few-shot learning with frozen language models," in *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 2021, pp. 200–212.

[35] F. Petroni, T. Rocktäschel, S. Riedel, P. S. H. Lewis, A. Bakhtin, Y. Wu, and A. H. Miller, "Language models as knowledge bases?" in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Association for Computational Linguistics, 2019, pp. 2463–2473.

[36] T. Schick and H. Schütze, "Few-shot text generation with pattern-exploiting training," *CoRR*, vol. abs/2012.11926, 2020.

[37] R. Shin, C. H. Lin, S. Thomson, C. Chen, S. Roy, E. A. Platanios, A. Pauls, D. Klein, J. Eisner, and B. V. Durme, "Constrained language models yield few-shot semantic parsers," 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*. Association for Computational Linguistics, 2021, pp. 7699–7715.

[38] Z. Jiang, F. F. Xu, J. Araki, and G. Neubig, "How can we know what language models know," *Trans. Assoc. Comput. Linguistics*, vol. 8, pp. 423–438, 2020.

[39] Qemu, "qemu/qemu," in *https://github.com/qemu/qemu*, 2023.

[40] "Cve-2020-7248," in *https://www.cvedetails.com/cve/CVE-2020-7248*, 2023.

[41] X. Hu, S. Yu, C. Xiong, Z. Liu, Z. Liu, and G. Yu, "P3 ranker: Mitigating the gaps between pre-training and ranking fine-tuning with prompt-based learning and pre-finetuning," in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022, pp. 1956–1962.

[42] Z. Zhong, D. Friedman, and D. Chen, "Factual probing is [MASK]: learning vs. learning to recall," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*. Association for Computational Linguistics, 2021, pp. 5017–5033.

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

[44] S. E. Ponta, H. Plate, A. Sabetta, M. Bezzi, and C. Dangremont, "A manually-curated dataset of fixes to vulnerabilities of open-source software," in *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada*. IEEE / ACM, 2019, pp. 383–387.

[45] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. ACM, 2016, pp. 785–794.

[46] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017, pp. 3146–3154.

[47] C. J. Burges, "From ranknet to lambdarank to lambdamart: An overview," *Learning*, vol. 11, no. 23-581, p. 81, 2010.

[48] "Pytorch," https://pytorch.org/, 2022.

[49] "Huggingface," https://huggingface.co/models, 2022.

[50] N. Ding, S. Hu, W. Zhao, Y. Chen, Z. Liu, H. Zheng, and M. Sun, "Openprompt: An open-source framework for prompt-learning," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, ACL 2022 - System Demonstrations, Dublin, Ireland, May 22-27, 2022*. Association for Computational Linguistics, 2022, pp. 105–113.

[51] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[52] Openssl, "Openssl project," in *https://github.com/openssl/openssl*, 2023.

[53] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "A c/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.

[54] Cvedetail., "Cvedetail," https://www.cvedetails.com, 2022.

[55] Z. Xu, B. Chen, M. Chandramohan, Y. Liu, and F. Song, "SPAIN: security patch analysis for binaries towards understanding the pain and pills," in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*. IEEE / ACM, 2017, pp. 462–472.

[56] C. Xu, B. Chen, C. Lu, K. Huang, X. Peng, and Y. Liu, "Tracking patches for open source software vulnerabilities," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 860–871.

[57] V. Le and H. Zhang, "Log parsing with prompt-based few-shot learning," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2023, pp. 2438–2449.

[58] Q. Huang, Y. Sun, Z. Xing, M. Yu, X. Xu, and Q. Lu, "API entity and relation joint extraction from text via dynamic prompt-tuned language model," *CoRR*, vol. abs/2301.03987, 2023.

[59] Q. Huang, Z. Yuan, Z. Xing, X. Xu, L. Zhu, and Q. Lu, "Prompt-tuned code language model as a neural knowledge base for type inference in statically-typed partial code," in *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 2022, pp. 79:1–79:13.

[60] Z. Liu, C. Chen, J. Wang, X. Che, Y. Huang, J. Hu, and Q. Wang, "Fill in the blank: Context-aware automated text input generation for mobile GUI testing," *CoRR*, vol. abs/2212.04732, 2022.

[61] X. Luo, Y. Xue, Z. Xing, and J. Sun, "PRCBERT: prompt learning for requirement classification using bert-based pretrained language models," in *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 2022, pp. 75:1–75:13.