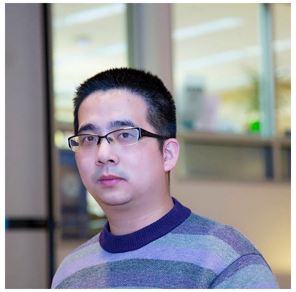
Dual Prompt-Based Few-Shot Learning for Automated Vulnerability Patch Localization

Junwei Zhang   Xing Hu   Lingfeng Bao   Xin Xia   Shanping Li




SANER 2024, Rovaniemi, Finland

The IEEE International Conference on Software Analysis, Evolution and Reengineering

What is a security vulnerability?

A vulnerability is an issue or inherent weakness that can result in a successful attack.

What is the CVE:

CVE, which stands for Common Vulnerabilities and Exposures, is a list of publicly disclosed network security vulnerabilities.

CVEs Received and Updated-14 August 2023 in CVEdetails.com [1]

Time Period	New CVEs Received by NVD	CVEs Analyzed by NVD
One Day	135	265
Last Week	809	1500
Last Month	2279	4358

NVD: National Vulnerability Database

Example of CVE with missing code commit link

Vulnerability Report

CVE ID: CVE-2020-7248
Disclosure Date: July, 03, 2019
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) {
```

Vulnerability Report

CVE ID: CVE-2020-7248
Disclosure 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
References:
MISC:https://github.com/openwrt/openwrt/commits/master

A 67 days window of opportunity

Challenges with Existing Approaches

Searching-based approaches

Searching **CVE-ID** or **CVE-related URLs**, **patch-related URLs**, **git-related URLs** in CVE description.

Learning-based approaches

Neural network based models: PatchScout [2]
PLMs-based models: VCMatch [3] , VulCurator [4]

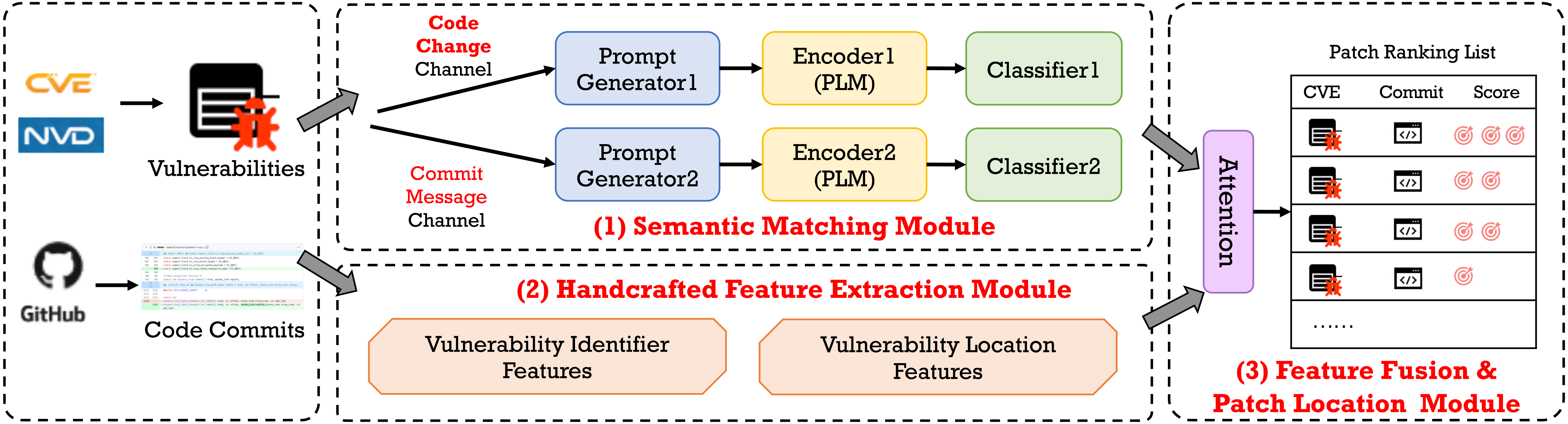
The Coverage Rate of the Patches Located for Disclosed Vulnerabilities in Searching-based Approaches

Approach	Coverage Rate
A1	8.52%
A2	6.53%
A3	12.48%
A4	19.61%
A1+A2+A3+A4	38.17%

A1: Searching CVE-ID or CVE-related URLs in commit message;
A2: Searching patch-related URLs in vulnerability descriptions;
A3: Searching git-related URLs in vulnerability descriptions;
A4: Searching bug-related keywords in vulnerability descriptions.

1. They cannot perform well, especially in data scarcity scenarios.
2. They are less effective in exploring semantic correlations between vulnerability descriptions and code commits.

Overview of PromVPat

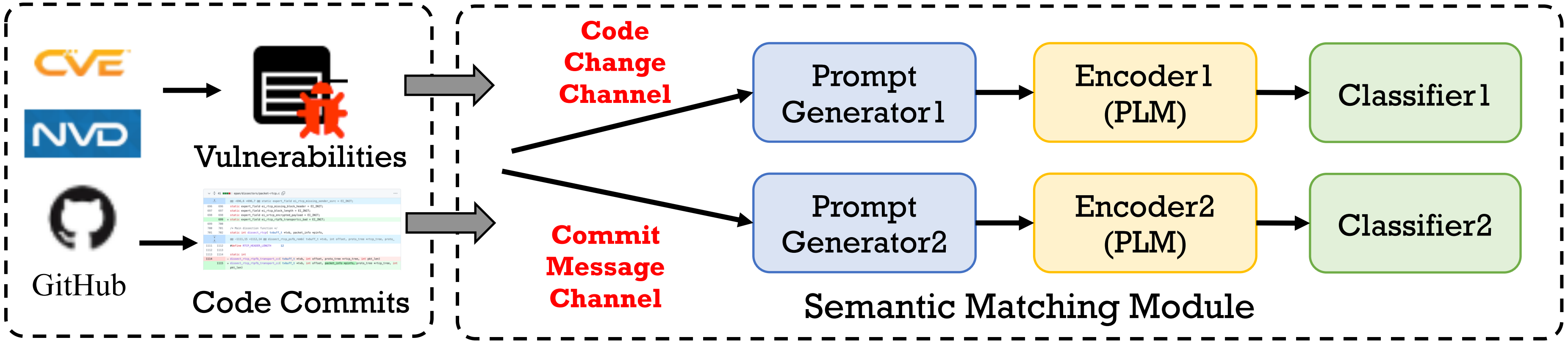


To alleviate the above two limitations:

1. Utilize **the prompt tuning method** to fine-tune PLMs

2. Propose a novel **dual prompt tuning channel** with two prompts

Module 1:Semantic Matching Module - Dual Prompt Tuning Channel



Code Change Prompt Generator 1:

The CVE $[x_1]$ is fixed by the code $[x_2]$. $[z]$

Commit Message Prompt Generator 2:

The CVE $[x_1]$ means $[x_3]$. Is it correct? $[z]$

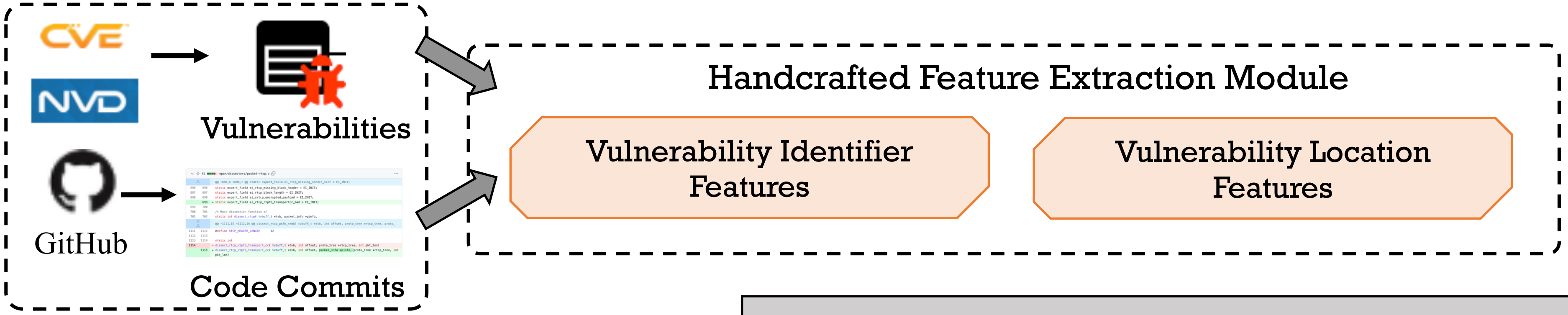
Encoder:

We adopt **CodeT5** as the encoder to generate the input representations. Initially, we freeze the CodeT5's parameters and derive the embeddings of the prompt tokens $P \in \mathbb{R}^{p \times d}$.

Classifier:

A SoftMax classifier uses the learned input text representation to determine the answer word distribution.

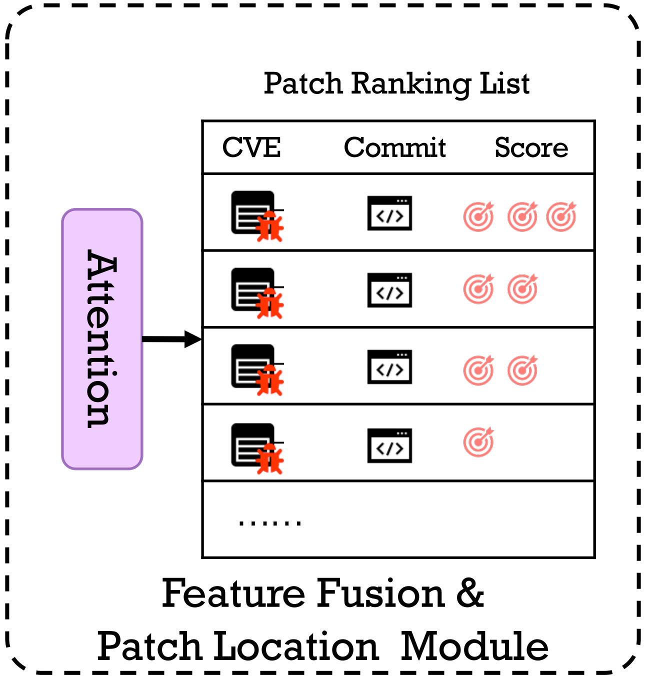
Module 2:Handcrafted Feature Extraction Module



Vulnerability Identifier Features
Number of CVE IDs, Bug IDs in commit messages
Whether the CVE IDs, Bug IDs, CWE IDs in commit messages
Number, Ratio of same words between CVE description and commit messages
Max, Sum, Average, Variance frequencies of same words between CVE descriptions and commit messages
Number, Ratio of same words between CVE descriptions and code changes
Max, Sum, Average, Variance frequencies of same words between CVE descriptions and code changes

Vulnerability Location Features
Number, Ratio of same file paths between CVE descriptions and commits
Number of file paths that is in commits but not mentioned in CVE descriptions
Number, Ratio of same files between CVE descriptions and commits
Number of files in commits but not mentioned in CVE descriptions
Number, Ratio of functions between CVE descriptions and commits
Number of functions in commits but not mentioned in CVE descriptions

Module 3: Feature Fusion & Patch Location Module



We apply the **dot-production attention mechanism** 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 $\hat{y}$.

$$\hat{y} = Softmax\left(MLP\left(tanh(\hat{C})\right)\right)$$

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

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

Baseline:

1. **Traditional classification models**: XGBoost, LightGBM
2. **Neural network based models**: PatchScout
3. **PLMs-based models**: VCMatch, VulCurator

Metrics:

1. **Recall@K**
2. **NDCG(Normalized Discounted Cumulative Gain)@K**

Dataset:

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

RQ1: Effectiveness on Patch Localization

Table 1. 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%

1. Our approach achieves the best performance regarding all metrics.
2. Traditional classifiers (i.e., XGBoost and LightGBM) are limited in locating the security patches.
3. PLMs-based models (i.e., VCMatch, VulCurator, and PromVPat) outperform the other baseline models.

RQ2: Effect of Different Prompt Tuning Designs

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%**

(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

1. PromVPat outperforms all its variants across four metrics.
2. The performance improvement of PromVPat over PromVPat-mess in Recall is less than the improvement over PromVPat-code.

RQ2: Effect of Different Prompt Tuning Designs

Evaluation Results of Different Prompt Templates

Channel Type	Prompt Templates	Metrics R@1	Metrics 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%

1. The choice of prompt templates significantly impacts our approach’s effectiveness.
2. PromVPat achieves the best performance in all metrics.

RQ2: Effect of Different Prompt Tuning Designs

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%

PromVPat-CodeBERT and PromVPat-GPT2 use CodeBERT and GPT2 to encode the input text, respectively.

1. PromVPat achieves superior results, outperforming its counterparts using PromVPat-CodeBERT and PromVPat-GPT.

RQ3: Effect of different handcrafted features for the proposed model?

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%**

1. Each category significantly boosts the performance of our approach.
2. The vulnerability identifier features contribute the most

RQ4: Effectiveness in Data Scarcity Scenarios

Evaluation Results in the Cross-Language Low-Resource Scenario

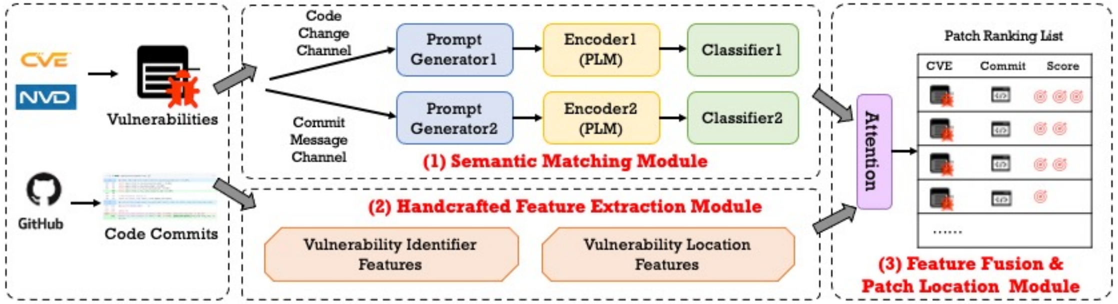
Method	Different Shots			
	1 Shot	4 Shots	8 Shots	16 Shots
Fine-Tuning	48.78%	49.39%	48.78%	52.44%
Prompt-Tuning	**51.21%**	**53.05%**	**53.66%**	**54.88%**

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%**

1. Prompt tuning achieves better performance than fine-tuning in all few-shot settings

Overview of PromVPat



To alleviate the above two limitations:
1. Utilize **the prompt tuning method** to fine-tune PLMs
2. Propose a novel **dual prompt tuning channel** with two prompts

RQ1: Effectiveness on Patch Localization

Table 1. 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%

1. Our approach achieves the best performance regarding all metrics.
2. Traditional classifiers (i.e., XGBoost and LightGBM) are limited in locating the security patches.
3. PLMs-based models (i.e., VCMatch, VulCurator, and PromVPat) outperform the other baseline models.

RQ2: Effect of Different Prompt Tuning Designs

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%**

(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

1. PromVPat outperforms all its variants across four metrics.
2. the performance improvement of PromVPat over PromVPat-mess in Recall is less than the improvement over PromVPat-code.

RQ2: Effect of Different Prompt Tuning Designs

Evaluation Results of Different Prompt Templates

[无标题]

Channel Type	emplates	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%

1. The choice of prompt templates significantly impacts our approach's effectiveness.
2. PromVPat achieves the best performance in all metrics.

Background Methodology **Evaluation** Summary Reference

- **RQ2: Effect of Different Prompt Tuning Designs**

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%

PromVPat-CodeBERT and PromVPat-GPT2 use CodeBERT and GPT2 to encode the input text, respectively.

1. PromVPat achieves superior results, outperforming its counterparts using PromVPat-CodeBERT and PromVPat-GPT.

Background Methodology **Evaluation** Summary Reference

- **RQ3: Effect of different handcrafted features for the proposed model?**

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%**

1. Each category significantly boosts the performance of our approach.
2. The vulnerability identifier features contribute the most

Background Methodology **Evaluation** Summary Reference

- **RQ4: Effectiveness in Data Scarcity Scenarios**

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%**

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%**

1. Prompt tuning achieves better performance than fine-tuning in all few-shot settings

Thanks for listening
Q & A

Contact E-mail: jw.zhang@zju.edu.cn

Code Link: https://zenodo.org/records/10520971

Reference

[1] https://www.cvedetails.com/

[2] Tan X, Zhang Y, Mi C, et al. Locating the security patches for disclosed oss vulnerabilities with vulnerability-commit correlation ranking[C]//Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. 2021: 3282-3299.

[3] Wang S, Zhang Y, Bao L, et al. Vcmatch: a ranking-based approach for automatic security patches localization for OSS vulnerabilities[C]//2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE, 2022: 589-600.

[4] Nguyen T G, Le-Cong T, Kang H J, et al. Vulcurator: a vulnerability-fixing commit detector[C]//Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2022: 1726-1730.