Assessing the Effectiveness of Vulnerability Detection via Prompt Tuning: An Empirical Study

Guilong Lu[†], Xiaolin Ju[†*], Xiang Chen[†*], Shaoyu Yang[†], Liang Chen[†], Hao Shen[†]
[†]*School of Information Science and Technology*, *Nantong University*, China
guil.lu@outlook.com, {ju.xl, xchencs, chenliang82}@ntu.edu.cn, {shaoyuyoung, shenhyc}@gmail.com

***Abstract*—In vulnerability detection approaches based on deep learning, fine-tuning with Pre-trained Language Models (PLMs) is a prevalent technique. Unfortunately, a natural gap exists between model pre-training tasks and vulnerability detection tasks due to different input formats, and the performance of fine-tuning relies on downstream dataset scales. Recently, prompt tuning has been used to alleviate these issues. However, it has not received enough attention in vulnerability detection. To assess the effectiveness of prompt tuning, we consider three classical vulnerability detection tasks: within-domain vulnerability detection, cross-domain vulnerability detection, and vulnerability type detection. Our empirical study considers three popular PLMs: CodeBERT, CodeT5, and CodeGPT. Then we use Devign, BigVul, and Reveal datasets as our experimental subjects. Our empirical results indicate that (1) compared to fine-tuning, prompt tuning can increase the accuracy of three tasks by an average of 42%, 38%, and 41%, respectively; (2) different prompt templates can have up to an 8% impact on accuracy; (3) in data scarcity scenarios, the superiority of prompt tuning over fine-tuning is more obvious. Our research demonstrates that using prompt tuning can help to achieve better performance in vulnerability detection tasks and is a promising research direction in the future.**

***Index Terms*—Prompt tuning, Vulnerability detection, Vulnerability type detection, Cross-domain vulnerability detection**

I. Introduction

Software vulnerabilities can be exploited by malicious attackers, thereby jeopardizing the system's confidentiality, integrity, and availability. For example, in 2021, a group exploited the ProxyLogon vulnerability [1] to steal internal data from Acer and publicly demanded a ransom of 50 million US dollars. Consequently, vulnerability detection has become increasingly important as it is critical to ensuring software security.

In previous studies, researchers proposed program analysis (PA)-based approaches to detect vulnerabilities [2]. For instance, RATS[1] can identify vulnerabilities, including buffer overflows and TOCTOU race conditions [3]. Cppcheck[2] used unique code analysis and focused on detecting undefined behavior and dangerous code structures [3]. Unfortunately, PA-based approaches rely on predefined patterns to identify vulnerabilities [4], and the predefined patterns need to be manually created by security experts, which can be time-consuming.

*Corresponding author
[1]https://github.com/andrew-d/rough-auditing-tool-for-security
[2]http://cppcheck.net/

Deep learning (DL)-based techniques can effectively mitigate this problem, especially with the fine-tuning approaches based on PLMs, which have become the mainstream approaches for vulnerability detection [5]. These PLMs, such as CodeBERT [6] and CodeT5 [7], learn from large-scale corpus to better understand underlying knowledge. Subsequently, fine-tuning the PLMs enables them to better adapt to vulnerability detection. For example, Hanif *et al.* [8] pre-trained a custom tokenization pipeline on real-world C/C++ code to train a Roberta model [9]. Then, this model was fine-tuned for vulnerability detection tasks. Similarly, Fu *et al.* [10] constructed a model with a BERT architecture [11] and pre-trained it, and then fine-tuned it for vulnerability detection at the line level. However, the inconsistent inputs and objectives of pre-training and fine-tuning make it difficult to fully explore the knowledge of PLMs [12], [13], leading to sub-optimal results for downstream tasks. Moreover, the effectiveness of using fine-tuning largely depends on the scale of the downstream data [14], [15].

Recently, prompt tuning has emerged as an advanced technology that has demonstrated promising performance in many language modeling tasks [16], [17]. Unlike fine-tuning, prompt tuning is a recent approach in natural language processing (NLP) that involves training a PLM on a small amount of task-specific data and a set of prompts, which are structured prompts that guide the model to generate task-specific outputs. Prompt tuning has been proven superior to fine-tuning in many NLP tasks, such as text classification [15] and emotion detection [18]. To our best knowledge, its effectiveness in vulnerability detection tasks has not been thoroughly explored.

Inspired by the success of prompt tuning in the field of NLP [15], [18], we explore whether prompt tuning is effective in vulnerability detection tasks. In our empirical study, we aim to answer the following research questions (RQs):

- RQ1: How effective is prompt tuning in vulnerability detection tasks?
- RQ2: How effective is prompt tuning in vulnerability type detection tasks?
- RQ3: How do different prompt templates affect the performance of prompt tuning?
- RQ4: How effective is prompt tuning in data scarcity scenarios?
- RQ5: How effective is prompt tuning for cross-domain vulnerability detection tasks?

To answer these RQs, we conduct experiments using fine-tuning and prompt tuning on three PLMs (i.e., CodeBERT, CodeT5, and CodeGPT) which achieve state-of-the-art performance in many code-related downstream tasks [19]–[21] and evaluate their performance on three datasets (i.e., Devign, BigVul, and Reveal) which are popular benchmark dataset. The results indicate that prompt tuning outperforms fine-tuning on these tasks. Moreover, the design of different prompt templates and verbalizers can also have an impact on performance.

The main contributions of our study are summarized as follows:

- We conduct the first analysis of the effectiveness of prompt tuning in the field of vulnerability detection tasks.
- We evaluate the effectiveness of prompt tuning using three PLMs (i.e., CodeBERT, CodeT5, and CodeGPT) and evaluate its performance on three datasets (i.e., Devign, BigVul, and Reveal).
- We investigate the effectiveness of prompt tuning using different prompt templates and discuss its performance in data scarcity scenarios.
- We summarize three implications of our study that can guide future research.

Our source code and experimental data are available at: *https://github.com/P-E-Vul/prompt-empircial-vulnerability*.

II. Background and Research Motivation

A. Vulnerability detection based on PLMs

The PLM-based vulnerability detection approaches utilize DL-based approaches to identify and detect security vulnerabilities in software [22]. These PLMs have been trained on vast corpus [6], [7]. During pre-training, they learn transferable and generic feature representations. Subsequently, PLMs can adapt and customize their knowledge representations for new tasks by fine-tuning using the data and labels of those tasks. Typically, these approaches involve five steps [23]: (1) Data collection and preprocessing: collect a large number of source code samples, including both vulnerable and non-vulnerable code, and preprocess them by removing comments, blank lines, etc.; (2) Feature extraction and representation: convert the source code into a format suitable for processing by deep learning models; (3) Model selection and training: choose an appropriate PLM, such as CodeGPT [24] or CodeBERT [6], as the base model, and fine-tune it for the vulnerability detection task using collected samples of vulnerable and non-vulnerable code; (4) Model evaluation: after training, evaluate the model using a test dataset; (5) Vulnerability detection: deploy the trained model in real-world scenarios to identify vulnerabilities in the target source code.

B. Prompt Tuning

Prompt tuning aims to convert the downstream task objectives of fine-tuning into pre-training tasks [9], [11]. As shown in Fig. 1(a), traditional fine-tuning involves inputting the [CLS] representation into a classifier for binary classification, predicting the presence or absence of vulnerabilities in a function, and requiring a significant amount of data for training. Prompt tuning generates a template with a special token [MASK] related to a given sentence, using methods such as manual definition [25], automatic search [26], and text generation [15]. This template, such as "The code is [MASK]" shown in Fig. 1(b), is concatenated with the original text to create the input for prompt tuning. This input is fed into a PLM to predict the distribution over label words on the [MASK] token position, such as "Good", "Vulnerable", "Bad" and so on. Then, the verbalizer outputs the final prediction using a mapping relationship that maps each label word to a class [15]. As illustrated in Fig. 1(b), label words such as "Vulnerable" and "Bad" correspond to code snippets with vulnerabilities, while "Good" corresponds to code snippets without vulnerabilities. The classes include "+" and "-", where "-" denotes the presence of vulnerabilities and "+" denotes the absence of vulnerabilities in this example.

Currently, prompt-tuning techniques are usually classified into two main categories within the academic community: hard prompts (discrete prompts) and soft prompts (continuous prompts). In the rest of this subsection, we provide details for each prompt type.

1) Hard Prompt: Hard prompt refers to the technique of modifying inputs of the models by adding natural language texts. The current construction methods of hard prompts mainly include manual construction, heuristic search, and generation methods [12]. In hard prompts, each added token is interpretable [14]. For example, the hard prompt template of the vulnerability detection task can be formulated as:

$$f_{\text{prompt}}(x) = \text{" The code } [X] \text{ is } [Z]\text{"} \tag{1}$$

where $[X]$ denotes the input code, and $[Z]$ denotes the label word that the model is required to predict, such as "vulnerable" or "non-vulnerable".

2) Soft Prompt: Hard prompts require designing a specific template for each task, as these templates consist of discrete, readable tokens, making it challenging to find the optimal template [15]. Moreover, even for the same task, different sentences may have their optimal templates [17]. Sometimes, even human-understandable, similar templates can produce significantly different model predictions [12]. Therefore, soft prompts have been proposed [15], [17], [27]. Unlike hard prompts, tokens in soft prompts are not fixed and discrete words in natural language, but virtual tokens, such as:

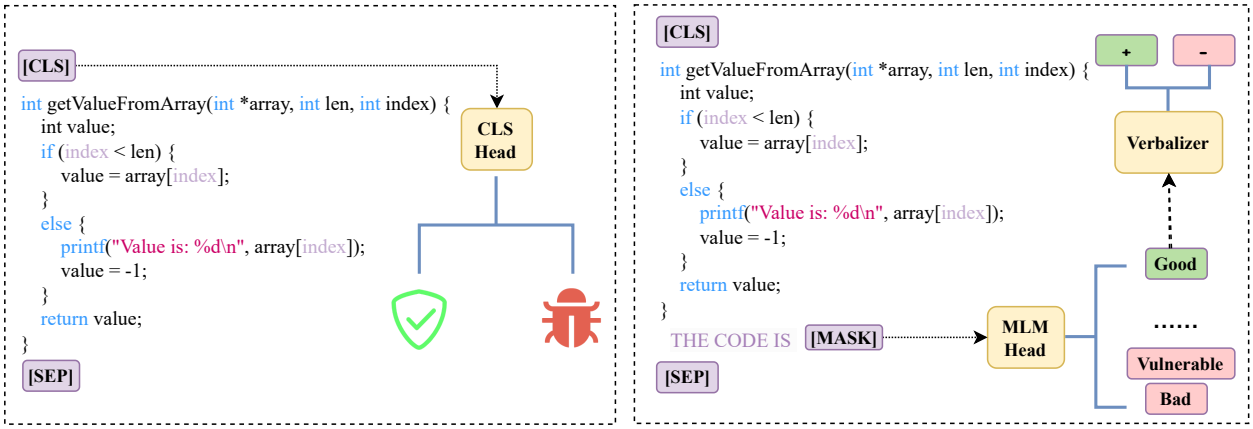
$$f_{\text{prompt}}(x) = \text{"}[SOFT][SOFT][X][SOFT][Z]\text{"} \tag{2}$$

where, $[SOFT]$ denotes a virtual token.

C. Research Motivation

In previous research, vulnerability detection based on PLMs was performed using fine-tuning and faced two main challenges.

- When fine-tuning the PLM for vulnerability detection tasks, the input is limited to source code, and the training objective is transformed into a classification problem. The



(a) Fine-tuning paradigm

(b) Prompt-tuning paradigm

Fig. 1: Explanation of the process of fine-tuning paradigm and prompt-tuning paradigm. The function in the figure is an example of CWE-119. [CLS] and [SEP] are special tokens in PLMs.

discrepancies between the inputs and objectives of pre-training and fine-tuning make it difficult to fully explore the knowledge of PLMs, leading to sub-optimal results for downstream tasks [12], [13];

- The effectiveness of using fine-tuning largely depends on the scale of the downstream data [14], [15].

Recent studies have shown that prompt tuning can mitigate the disadvantages of traditional fine-tuning, making models more adaptable to different downstream tasks [15], [18].

Regarding the first challenge, prompt tuning uses task-specific prompts to guide the model-tuning of PLMs, which can bridge the gap between fine-tuning and pre-training.

Regarding the second challenge, compared to traditional fine-tuning, prompt tuning does not require fine-tuning all the parameters of PLMs but only needs to train a small number of parameters related to the task. This can reduce the number of required training parameters, thus reducing the risk of overfitting and improving the model's generalization ability.

However, most current research is mainly limited to NLP tasks [12]. In the vulnerability detection task, the advantages of prompt tuning have not been proven effective yet. Therefore, in this paper, we aim to evaluate whether prompt tuning outperforms fine-tuning in vulnerability detection comprehensively.

III. Research Questions

This section outlines the motivations behind five research questions. Our goal is to provide answers to these RQs by conducting an extensive experimental evaluation:

RQ1: How effective is prompt tuning in vulnerability detection tasks?

Motivation: Vulnerability detection tasks aim to identify whether a function contains vulnerabilities. Correctly identifying such vulnerabilities is crucial for improving system security [28]. By studying the effectiveness of prompt tuning and fine-tuning in vulnerability detection tasks, we can better understand the applicability of two techniques and provide better solutions for vulnerability detection.

RQ2: How effective is prompt tuning in vulnerability type detection tasks?

Motivation: Vulnerability type detection tasks aim to classify and identify vulnerabilities in software programs. High-quality detection of vulnerability types is crucial for locating and fixing vulnerabilities [29]. We aim to study the effectiveness of prompt tuning in vulnerability type detection, which can also provide valuable insights for other multi-classification tasks.

RQ3: How do different prompt templates affect the performance of prompt tuning?

Motivation: During the prompt tuning, prompt templates can serve as guidance to help the model better learn the task features and data features. However, if the selected prompt templates are inappropriate or poorly designed, the generated prompts may contain bias or inaccuracies, which can negatively impact the performance of the model [12]. Currently, there is insufficient understanding of the effectiveness of different prompt templates in vulnerability detection tasks, and further research is needed to help researchers better apply and optimize prompt tuning techniques.

RQ4: How effective is prompt tuning in data scarcity scenarios?

Motivation: Recent research has shown that fine-tuning performance is highly dependent on the size of the downstream task dataset [30], [31]. However, dataset scarcity is a common issue in the context of vulnerability detection tasks. We design this RQ to investigate the effectiveness of prompt tuning in data scarcity.

RQ5: How effective is prompt tuning for cross-domain vulnerability detection tasks?

Motivation: Cross-domain vulnerability detection refers to detecting vulnerabilities within a domain or application by utilizing data, approaches, or models from different domains

or applications. Cross-domain vulnerability detection is a critical problem since models need to generalize to different domains and scenarios. We design this RQ to investigate the performance of prompt tuning for cross-domain vulnerability detection.

IV. Case Study Design

A. Evaluation Subjects

1) Vulnerability detection tasks: To empirically evaluate the performance of prompt tuning in vulnerability detection tasks, we select three popular benchmark datasets (i,e., Devign [32], BigVul [33], and Reveal [34]).

- **Devign** [32]. The Devign dataset was manually labeled and sourced from two open-source C projects: FFmpeg and Qemu. It includes 10,067 vulnerable functions and 12,294 non-vulnerable functions.
- **BigVul** [33]. The BigVul dataset was sourced from over 300 open-source C/C++ projects on GitHub, covering 91 distinct vulnerability types listed in the Common Vulnerabilities and Exposures (CVE) database from 2002 to 2019. The dataset comprises 10,547 vulnerable functions and 179,299 non-vulnerable functions.
- **Reveal** [34]. The Reveal dataset was sourced from two open-source projects: Linux Debian Kernel and Chromium. It comprises 16,505 vulnerable functions and 1,664 non-vulnerable functions.

We use Table I to present the statistical information for these datasets, including the total number of samples, the number of samples with vulnerabilities, the number of samples without vulnerabilities, and the ratio of samples with vulnerabilities to the total number of samples.

TABLE I: Statistics of the dataset used in vulnerability detection tasks

Dataset	Samples	Vul	Non-vul	Vul Ratio(%)
Devign [32]	22361	10067	12294	45.02
BigVul [33]	179299	10547	168752	5.88
Reveal [34]	18169	1664	16505	9.16

We select these three datasets for the following reasons: (1) all three datasets contain real-world projects and vulnerabilities; (2) most models in related literature evaluate their performance on these three datasets, such as IVDetect [35] and AMPLE [36]; and (3) the BigVul dataset includes annotations on vulnerability types, which are essential for addressing our RQ2.

2) Vulnerability type detection tasks: Our study is focused on exploring the application of prompt tuning in vulnerability type detection tasks rather than prioritizing the evaluation of numerous vulnerability types. As a result, we select five vulnerability types that rank among the top five on the CWE official website[3], as they are commonly found in practical software development and are widely recognized as posing potential threats to network security. For this task, we utilize the dataset that has been extracted from BigVul [33].

In Table II, we present the statistical information of the dataset, including details on the types of vulnerabilities, their corresponding names, the respective quantities of each vulnerability type, and their proportion relative to the overall dataset size.

TABLE II: Description of vulnerability types used in vulnerability type detection tasks

Rank	CWE-ID	Name	Numbers	Type Ratio(%)
1	CWE-79	Cross-site Scripting	9	1.07
2	CWE-787	Out-of-bounds Write	44	5.28
3	CWE-20	Improper Input Validation	228	27.34
4	CWE-125	Out-of-bounds Read	107	12.83
5	CWE-119	Improper Restriction	446	53.48

B. Performance Measures

1) Vulnerability detection tasks: For vulnerability detection tasks, similar to AMPLE [36], we use the following four widely-used performance measures in our evaluation:

Accuracy. $Accuracy$ is a common metric used in binary classification tasks to measure the ability of the model to identify vulnerable code. It is defined as the ratio of correctly classified samples to the total number of samples in the dataset. The formula is defined as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

where TP refers to the number of samples that the model correctly predicts as having vulnerabilities among those that actually have vulnerabilities, TN refers to the number of samples that the model correctly predicts as not having vulnerabilities among those that do not have vulnerabilities, FP refers to the number of samples that the model incorrectly predicts as having vulnerabilities among those that do not have vulnerabilities, FN refers to the number of samples that the model incorrectly predicts as not having vulnerabilities among those that actually have vulnerabilities.

Precision. $Precision$ measures the proportion of actual positive samples (i.e., containing vulnerabilities) among samples predicted by the model as positive. The formula is defined as follows:

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

Recall. $Recall$ measures the ability of a model to detect all actual samples containing vulnerabilities, i.e., the proportion of actual samples containing vulnerabilities that are correctly predicted as positive by the model. The formula is defined as follows:

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

[3]https://cwe.mitre.org/

F1 score. $F1\ score$ represents a balance between precision and recall and is calculated as the harmonic mean of the two. The formula is defined as follows:

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{6}$$

2) Vulnerability type detection tasks: The categories for multi-class classification in the dataset are imbalanced, with the proportions shown in Table II, it does not imply that less represented vulnerability types are less important. For example, CWE-79, which has only nine instances in the dataset, is ranked first on the CWE official website. Therefore, for multi-class classification, we use accuracy, weighted F1 Score, and macro F1 Score as the evaluation metric.

Weighted F1 Score. $Weighted\ F1\ Score$ (w-F1) is calculated by taking a weighted average of the F1 Score for each class, where the weights are based on the relative frequency of each class in the dataset. The formula is defined as follows:

$$Weighted\ F1\ score = \sum_{i=0}^{m} w_i * (F1\ score)_i \tag{7}$$

where m is the number of classes, w_i is the weight for class i (calculated as the ratio of the number of samples in class i to the total number of samples).

Macro F1 Score. $Macro\ F1\ Score$ (m-F1) is a metric for evaluating the performance of multi-class classification models. It calculates the F1 score for each class and takes the average of all F1 scores. The formula is defined as follows:

$$Macro\ F1\ score = \frac{1}{n} * \sum_{i=0}^{n-1} (F1\ score)_i \tag{8}$$

where n is the number of categories. In the vulnerability type detection dataset used in our study, there are five types of vulnerabilities, so n is set to 5.

C. Pre-trained Language Models

We choose three widely used PLMs: CodeBERT [6], CodeT5 [7], and CodeGPT [24] which represent three different Transformer-based architecture (i.e., Encoder-only, Encoder-Decoder, and Decoder-only) since they have been trained on large-scale code corpus and shown promising performance in code-related downstream tasks [19]–[21]. We introduce them as follows:

CodeBERT [6] is a PLM based on the BERT (Bidirectional Encoder Representation from Transformers) [11] which is an encoder-only Transformer architecture. CodeBERT is pre-trained on CodeSearchNet corpus consisting of NL-PL (natural language and programming language) information which has 125 million parameters.

CodeT5 [7] is a PLM based on the T5 (Text-to-Text Transfer Transformer) [37] which is an encoder-decoder Transformer architecture, designed specifically for handling conversion tasks between NL and PL. CodeT5-small has 60 million parameters, and CodeT5-base has 220 million parameters.

CodeGPT [24] is an autoregressive PLM based on the GPT-2 (Generative Pre-Trained Transformer) [38] which is a decoder-only Transformer architecture. During the pre-training process of the CodeGPT model, a large amount of code corpus is used, enabling the model to adapt code-related tasks. CodeGPT has 124 million parameters.

D. Implementation Details

To ensure the fairness of the experiments, we use the same data split for all approaches. We randomly divide the dataset into non-overlapping training, validation, and testing sets at a ratio of 8:1:1. In the prompt tuning experiments, we utilize OpenPrompt [39] to construct various prompt templates and verbalizers. During the training process, we use a consistent number of 20 epochs for both prompt tuning and fine-tuning.

In our empirical study, we utilize PLMs and their corresponding tokenizers which are loaded from HuggingFace[4] repository. We set identical hyperparameters for fine-tuning and prompt tuning, e.g., the optimizer is AdamW [40], the learning rate is 5e-5, the lr scheduler type is linear, etc.

All of our investigations are conducted on a server equipped with an NVIDIA GeForce RTX 4090.

V. Result Analysis

A. RQ1: Effectiveness of prompt tuning in vulnerability detection tasks.

Method. To answer RQ1, we use the prompt tuning for vulnerability detection on CodeBERT, CodeT5, and CodeGPT. Our primary baseline is fine-tuning paradigm. For the CodeT5 model, we utilize the CodeT5-base and CodeT5-small to explore the impact of PLM size on prompt tuning performance. We conduct experiments on three commonly used vulnerability datasets (i,e., Devign [32], BigVul [33], and Reveal [34]).

Result. Table III compares prompt tuning and fine-tuning in different models. The results show that prompt tuning consistently performs better than traditional fine-tuning. Taking CodeT5-base with the best performance on the Devign dataset as an example, compared to fine-tuning, prompt tuning improves the accuracy, precision, recall, and F1 score by 0.64%, 8.42%, 2.18%, and 5.12%, respectively. These results indicate that PLMs can learn richer knowledge and have a more vital ability to extract contextual semantic information with input texts under prompt tuning.

Finding 1: In vulnerability detection tasks, prompt tuning can outperform fine-tuning in most cases. Taking CodeT5-small as an example, on average, compared to fine-tuning, prompt tuning has achieved a 3.03% improvement in accuracy, a 27.75% improvement in precision, a 5.63% improvement in recall, and a 15.21% improvement in F1 score.

B. RQ2: Effectiveness of prompt tuning in vulnerability type detection tasks.

Method. To answer RQ2, we conduct experiments on prompt tuning and fine-tuning for vulnerability type detection on three models. We select the top five most common vulnerability types in the CWE ranking (details can be found in Table II).

[4]https://huggingface.co/models

TABLE III: Comparison results of prompt tuning and fine-tuning on three datasets. Notice the bold font indicates better performance.

Model	Methods	Devign				BigVul				Reveal			
		Accuracy	Precision	Recall	F1 score	Accuracy	Precision	Recall	F1 score	Accuracy	Precision	Recall	F1 score
CodeBERT	Fine-tuning	61.71	56.70	50.29	53.30	94.24	47.90	31.99	38.36	88.21	40.63	33.75	36.87
	Prompt tuning	**62.99**	**61.08**	**52.27**	**56.33**	**97.98**	**91.55**	**70.50**	**79.66**	**89.05**	**43.13**	**34.31**	**38.21**
CodeT5-small	Fine-tuning	62.13	57.45	54.54	55.95	95.13	55.13	68.98	61.29	85.22	40.91	34.16	37.23
	Prompt tuning	**63.35**	**62.41**	**55.30**	**58.64**	**98.13**	**93.13**	**74.77**	**82.94**	**89.46**	**43.20**	**36.08**	**39.32**
CodeT5-base	Fine-tuning	63.56	58.55	54.97	56.70	96.01	50.94	**84.65**	63.60	85.92	41.13	36.19	38.50
	Prompt tuning	**63.97**	**63.48**	**56.17**	**59.60**	**98.20**	**93.39**	76.45	**84.07**	**87.11**	**43.75**	**36.46**	**39.77**
CodeGPT	Fine-tuning	62.49	57.16	51.36	54.10	95.13	49.62	60.10	54.36	84.29	39.16	34.01	36.40
	Prompt tuning	**63.03**	**59.01**	**62.88**	**60.88**	**97.42**	**82.51**	**69.04**	**75.17**	**87.42**	**40.64**	**35.57**	**37.93**

Result. Table IV presents the comparison results of two paradigms on the vulnerability type detection test set using three PLMs. We can find that prompt tuning performs better than fine-tuning. Taking CodeT5, which has the best prompt tuning performance, as an example, its w-F1 is 14.96% higher than that of fine-tuning. This result demonstrates that prompt tuning can extract knowledge from PLM more comprehensively and effectively by converting the classification task into the same cloze form as the pre-training stage.

TABLE IV: In vulnerability type detection tasks, performance comparison results between prompt tuning and fine-tuning.

Model	Methods	Accuracy	m-F1	w-F1
CodeBERT	Fine-tuning	53.12	44.19	56.13
	Prompt tuning	**63.40**	**55.77**	**65.26**
CodeT5	Fine-tuning	60.29	46.54	65.18
	Prompt tuning	**73.80**	**52.22**	**74.93**
CodeGPT	Fine-tuning	54.36	43.20	50.11
	Prompt tuning	**61.90**	**48.61**	**58.13**

Finding 2: In vulnerability type detection tasks, compared to fine-tuning, prompt tuning achieves 18.54%, 17.03%, and 15.74% improvement on average in accuracy, m-F1, and w-F1, respectively.

C. RQ3: Impact of Different Prompts.

1) Hard Prompt vs. Soft Prompt: **Method.** To answer this question, we establish seven distinct prompt template rules based on the location of the prompt template, as shown in Table V. The content within prompt can be replaced with a corresponding, human-readable natural language description. Table VI shows the seven distinct templates designed by us.

TABLE V: Prompt templates and their design rules

No.	Prompt template design rules
1	[X] prompt [Z]
2	[X] [Z] prompt
3	prompt [X] [Z]
4	prompt [X] prompt [Z]
5	[X] prompt [Z] prompt
6	prompt [X] [Z] prompt
7	prompt [X] prompt [Z] prompt

Result. By comparing the performance of seven different prompt templates in Table VI, we find that the design of prompt templates does have a significant impact on the performance of vulnerability detection models. For instance, when using the first prompt template "[X] the code is [Z]", the accuracy, precision, recall, and F1 score reached 62.59%, 60.13%, 62.24%, and 61.36%, respectively. However, when using the second prompt template, these metrics increased to 64.86%, 61.31%, 62.88%, and 62.08%, respectively. Moreover, by comparing the first and fourth prompt templates, which only differ in the order of tokens, we find that the fourth template achieved an accuracy of 64.47%. There is a 1.88% discrepancy between the two different prompt templates.

In addition, when using the same token order but different types of prompts, such as hard prompts and soft prompts, the results of vulnerability detection are also influenced. For example, when using the first token order, the accuracy of hard prompts and soft prompts reaches 62.59% and 62.22%, respectively.

2) Different Verbalizers: **Method.** To answer this question, we select labels associated with the vulnerability detection task as verbalizers, as displayed in Table VII. We investigate the impact of different verbalizers on the performance of prompt tuning. To comprehensively study the impact of verbalizers, we further varied the number of verbalizers. Specifically, we studied the cases where the number of verbalizers was set to 1 and 2, respectively.

Result. The results are presented in Table VII, which show that when setting the verbalizer to "vulnerable" and "non-vulnerable", the model achieved accuracy, precision, recall, and F1 score of 64.47%, 63.29%, 58.32%, and 60.69%, respectively. However, when setting the verbalizer to "good" and "bad", these metrics are 63.85%, 61.43%, 51.87%, and 56.24%, respectively. This indicates that different verbalizers can have some impact on the performance of prompt tuning. Moreover, we find that when the "+" and "-" signs of verbalizers are different, the effectiveness of prompt tuning is also affected. For instance, when "+" and "-" are mapped to "secure" and "insecure", respectively, the model achieved an accuracy of 63.37%. However, when the signs are swapped, but the verbalizers are the same, the accuracy increased to 65.47%. In addition, the results demonstrate that the number

TABLE VI: Comparison of the impact of different prompt templates on the performance of prompt tuning. The model is fixed to CodeT5-base, the verbalizer is fixed to "vulnerable, non-vulnerable", and the dataset is fixed to Devign. The bold font indicates better performance.

	Hard prompt	Soft prompt	Accuracy		Precision		Recall		F1 score	
			hard	soft	hard	soft	hard	soft	hard	soft
1.	[X] The code is [Z]	[X] [soft] [soft] [soft] [Z]	**62.59**	62.22	**60.13**	59.85	**62.24**	52.96	**61.36**	56.19
2.	[X] [Z] is the result	[X] [Z] [soft] [soft] [soft]	**64.86**	63.17	**61.31**	59.87	**62.88**	59.20	**62.08**	59.53
3.	Code: [X] [Z]	[soft] [soft] [X] [Z]	**63.03**	62.88	**60.34**	58.75	**62.49**	63.36	**61.40**	60.96
4.	The code [X] is [Z]	[soft] [soft] [X] [soft] [Z]	**64.47**	63.10	**63.28**	60.37	**58.32**	56.32	**60.69**	58.27
5.	[X] It is a [Z] code	[X] [soft] [soft] [soft] [Z] [soft]	**64.12**	64.05	**64.21**	60.50	48.80	**61.76**	55.45	**61.12**
6.	Code: [X] [Z] is the result	[soft] [soft] [X] [Z] [soft] [soft] [soft]	**62.90**	62.37	**63.01**	59.75	**54.60**	54.40	**58.50**	56.95
7.	The [X] is a [Z] code	[soft] [X] [soft] [soft] [Z] [soft]	**64.34**	63.32	**63.72**	59.62	**61.90**	61.44	**62.79**	60.52

TABLE VII: The impact of different verbalizers on the performance of prompt tuning. The prompt template is fixed to "The code [X] is [Z]". The model is fixed to CodeT5-base, and the dataset is fixed to Devign. The bold font indicates better performance.

Numbers	Verbalizer		Accuracy	Precision	Recall	F1 score
	+	-				
1	vulnerable	non-vulnerable	64.47	63.29	58.32	60.69
	secure	insecure	63.37	61.57	53.53	57.26
	good	bad	63.85	61.43	51.87	56.24
	bad	good	65.06	63.39	52.59	57.49
	insecure	secure	**65.47**	63.28	56.30	59.59
	non-vulnerable	vulnerable	63.59	61.52	**60.01**	**60.75**
2	vulnerable/insecure	non-vulnerable/secure	65.39	**64.19**	55.51	59.95
	good/secure	bad/insecure	65.10	65.13	55.26	59.79

of verbalizers also affects the performance of prompt tuning.

Finding 3: In different prompt types, we obtain 28 groups of results. Among them, 26 groups show that hard prompts outperformed soft prompts. Moreover, performance can be improved by using verbalizers that are more relevant to downstream tasks (e.g., "vulnerable"), and using two verbalizers instead of one can increase the accuracy and F1 score by 1.47% and 2.30%, respectively.

D. RQ4: Effectiveness of prompt tuning in data scarcity scenarios.

Method. To answer RQ4, we simulate zero-shot and few-shot scenarios by controlling the size of the training set to explore the ability of prompt tuning in these scenarios. To be specific, we establish five different scenarios for training shots, including 0 shot, 16 shots, 32 shots, 64 shots, and 128 shots. In the case of 0 shot, no tuning data was involved. The fine-tuning model directly generated target labels using the test data, while the prompt-tuning model predicted label words. To eliminate any randomness in the data selection process, we generate each subset ten times using different seeds and conduct four runs on each subset. The reported results are the average of these runs.

Result. In Table VIII, we present the accuracy and F1 score for five different prompt tuning and fine-tuning settings. Comparing these results with those in Table III, we observe a significant drop in performance. This is reasonable since PLMs require specific task data to perform well. In fact, for the CodeT5 and CodeGPT models, even failed to converge in the zero-shot scenario due to the lack of training data.

However, the experimental results demonstrate that prompt tuning still outperforms fine-tuning in data scarcity scenarios. For example, in the CodeBERT model, when the training set sizes are zero-shot, 16 shots, 32 shots, 64 shots, and 128 shots, the F1 score of prompt tuning was improved by 18.27%, 20.35%, 4.77%, 0.45%, and 0.96%, respectively, compared to fine-tuning. These findings suggest that prompt tuning can help the model better learn domain knowledge, even in data scarcity scenarios.

Finding 4: In data scarcity scenarios, prompt tuning outperforms fine-tuning at most 8.40% in accuracy and 34.10% in F1 score. Moreover, the extent of improvement provided by prompt tuning increases as the size of the training set decreases.

E. RQ5: Effectiveness of prompt tuning in cross-domain vulnerability detection tasks.

Method. To answer RQ5, we conduct cross-domain experiments on three different datasets, resulting in six sets of experiments comparing the performance of prompt tuning and fine-tuning. The three datasets comprise different projects, which can demonstrate the generalization ability of prompt tuning. We introduce the item composition of the datasets when describing them.

Result. Table IX shows that except for the case where the source domain is BigVul and the target domain is Reveal, prompt tuning outperforms fine-tuning in terms of both accu-

TABLE VIII: Comparison results of the performance between prompt tuning and fine-tuning under low-resource settings. "-" indicates that the model fails to converge due to extremely limited training data. The prompt template is fixed to "The code [X] is [Z]". The verbalizer is fixed to "vulnerable, non-vulnerable", and the dataset is fixed to Devign. The bold font indicates better performance.

Model	Methods	Zero shot		16 shots		32 shots		64 shots		128 shots	
		Accuracy	F1 score	Accuracy	F1 score	Accuracy	F1 score	Accuracy	F1 score	Accuracy	F1 score
CodeBERT	Fine-tuning	47.63	30.10	49.91	34.83	51.13	47.92	52.59	50.83	52.96	51.26
	Prompt tuning	**50.64**	**35.62**	**51.81**	**41.92**	**52.01**	**50.20**	**53.09**	**51.06**	**53.54**	**51.75**
CodeT5-small	Fine-tuning	-	-	47.26	43.81	48.16	48.95	50.01	53.19	52.78	53.49
	Prompt tuning	-	-	**49.30**	**46.99**	**50.76**	**51.26**	**51.34**	**53.78**	**53.43**	**54.77**
CodeT5-base	Fine-tuning	-	-	50.90	47.56	51.40	51.70	51.62	53.20	53.93	54.18
	Prompt tuning	-	-	**52.86**	**49.09**	**52.17**	**52.63**	**52.56**	**54.01**	**54.18**	**55.60**
CodeGPT	Fine-tuning	-	-	43.26	30.29	45.11	33.03	47.33	36.57	50.11	38.23
	Prompt tuning	-	-	**46.91**	**40.61**	**47.86**	**41.77**	**49.16**	**43.26**	**51.02**	**44.61**

racy and F1 score. Notably, when the source domain is Devign and the target domain is Reveal, prompt tuning achieves a 289.04% improvement in accuracy compared to fine-tuning.

TABLE IX: Comparison of the performance between prompt tuning and fine-tuning in cross-domain settings. The bold font indicates better performance.

Source->Target	Methods	Accuracy	F1 score
Devign->Reveal	Fine-tuning	22.91	19.77
	Prompt tuning	**89.13**	**25.38**
Devign->BigVul	Fine-tuning	21.89	11.74
	Prompt tuning	**25.72**	**11.88**
Reveal->Devign	Fine-tuning	55.60	5.49
	Prompt tuning	**55.89**	**9.74**
Reveal->BigVul	Fine-tuning	90.23	5.23
	Prompt tuning	**91.46**	**6.52**
BigVul->Devign	Fine-tuning	56.40	**4.33**
	Prompt tuning	**56.55**	2.15
BigVul->Reveal	Fine-tuning	86.93	18.63
	Prompt tuning	**88.20**	**24.72**

Finding 5: In cross-domain vulnerability detection tasks, we obtained 12 groups of results. Among them, 11 groups show that prompt tuning outperformed fine-tuning. On average, compared to fine-tuning, prompt tuning achieves a 57.53% improvement in accuracy and a 36.14% improvement in F1 score.

VI. Discussion

A. Implications of Our Empirical Study

1) Evaluating the Effectiveness of Prompt Tuning for Different PLMs: Based on our finding 1, 2, and 5, we observe that the improvement in the effectiveness of prompt tuning varies when using different PLMs. Currently, despite the availability of a large number of PLMs [7], [20], [24], [41], there is a lack of systematic comparison to evaluate the impact of different PLMs on prompt tuning. Therefore, exploring how to choose the appropriate PLM for prompt tuning is an important direction for future research.

2) Application of Prompt Tuning to other Tasks related to Software Vulnerability.: Based on our finding 1, 2, 4, and 5, we observe that prompt tuning outperforms fine-tuning in all three vulnerability detection tasks, especially when data resources are scarce. Therefore, comparing prompt tuning and fine-tuning to other more comprehensive and specific tasks related to software vulnerability is an important direction for future research (such as vulnerability assessment and prioritization [42]).

3) Incorporating Domain-Specific Knowledge to Prompt Tuning: Based on our finding 3, we observe that incorporating domain-specific knowledge into prompt template design can significantly improve model performance. Code structure information has been demonstrated to be effective in many DL models for code-related tasks [34], [43]. Therefore, considering incorporating code structure information into the design of prompt tuning to improve the performance of vulnerability detection is an important direction for future research.

B. Threats to Validity

Threats to the construct validity. Our experiments show that prompt tuning can enhance the performance of PLMs on vulnerability detection tasks. However, the prompt templates used in this study may not represent the most effective approach. To alleviate this threat, we develop a set of template rules and examine the influence of hard prompts and soft prompts on model performance.

Threats to the internal validity. During the evaluation of our models, there may be a certain degree of variability in the results of each run. For example, different runs of the same prompt template in RQ3 may yield other metrics. To alleviate this threat, we run each prompt template five times and average the results.

Threats to the external validity. The results may only apply to the specific models used and may not be generalizable to other models. To alleviate this threat, we employ three models: CodeBERT, CodeT5, and CodeGPT. These models have different architectures: encoder-only, encoder-decoder, and decoder-only, respectively.

Threats to the conclusion validity. The experimental results in this study are based on a limited number of datasets,

and their labels may not be entirely accurate. This could potentially introduce some bias into our results. To alleviate this threat, we select several widely used and high-quality datasets.

VII. Related Work

A. DL-based Vulnerability Detection

Traditional DL-based vulnerability detection approaches require the manual and time-consuming collection of metrics as features [44]. To address this issue, various deep learning-based approaches have been proposed to automatically learn vulnerability patterns from data [32], [34], [35].

Recurrent Neural Network-based (RNN-based) architectures are used to learn the syntax and semantics of source code automatically. For instance, Dam *et al.* [45] proposed a Long Short-Term Memory-based (LSTM-based) architecture to learn the syntax and semantics of source code automatically. However, the RNN-based approaches typically assume that the source code is a sequence of tokens and do not consider the graph structure of the source code, which can lead to inaccurate predictions.

Therefore, to better utilize the structure information of code, many approaches have abstracted code as graphs and used Graph Neural Networks (GNNs) to learn graph features [34], [43]. For example, Reveal [34] used Gated Graph Neural Network (GGNN) to process multiple directed graphs generated from source code. Similarly, MVD [43] used Program Dependence Graph (PDG) to represent code, performed program slices and used Flow-Sensitive Graph Neural Networks (FS-GNN) to detect vulnerabilities.

Prompt tuning has recently emerged as a new paradigm for enhancing downstream task performance by optimizing prompts to better adapt the model to specific task requirements and domains [12]. However, by analyzing existing work, we find whether using prompt tuning can improve the performance of vulnerability detection tasks has not been thoroughly investigated and our study wants to fill this gap.

B. Prompt Tuning

Jiang *et al.* [46] proposed a mining-based approach that automatically found a template given a set of training inputs x and outputs y. Yuan *et al.* [47] proposed a paraphrasing-based approach that replaced it with phrases in a thesaurus, returned a set of other candidate prompts, and then selected the one that achieved the highest training accuracy on the target task. Wallace *et al.* [48] applied a gradient-based search to actual tokens to find short sequences that could trigger a PLM to generate the desired target prediction.

Previous research has focused on applying prompt tuning in NLP. Currently, some studies have applied prompt turning to software engineering. Wang *et al.* [49] used prompt tuning to three downstream tasks in the field of software engineering, demonstrating the effectiveness of prompt tuning in this domain. Li *et al.* [50] used the prompt tuning to predict the severity of vulnerabilities and exploitability features based on vulnerability descriptions.

However, prompt tuning has not been applied to vulnerability detection in existing research. In this study, we want to perform a comprehensive analysis of prompt tuning for within-domain vulnerability detection, cross-domain vulnerability detection, and vulnerability type detection.

VIII. Conclusion and Future Work

Fine-tuning the PLMs has become a widely-used technique for vulnerability detection [8]. However, there is a natural gap between model pre-training tasks and vulnerability detection tasks due to differences in input formats, which can impact downstream task performance [12]. Prompt tuning has been shown to alleviate this issue and has demonstrated strong performance in many NLP tasks [15], [18]. Inspired by this, we conduct empirical evaluations to determine whether prompt tuning is superior to fine-tuning for vulnerability detection tasks. We perform large-scale studies on three PLMs (i.e., CodeBERT, CodeT5, and CodeGPT) across three vulnerability datasets (i.e., Devign, BigVul, and ReVeal). Our experiments indicate that prompt tuning outperforms fine-tuning in within-domain vulnerability detection, cross-domain vulnerability detection, and vulnerability type detection, particularly in data scarcity scenarios. Furthermore, our results show that different prompt types and verbalizers can have an impact on F1 score at most 7.34%.

Our future plans involve developing more suitable prompt templates that can enable the model to acquire domain-specific knowledge in a more effective manner. Additionally, we aim to expand our investigation of prompt tuning performance to a wider range of vulnerability datasets and PLMs.

References

[1] H. Gabriel, "Analỳza a demonstrace zranitelnosti proxylogon," B.S. thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2022.

[2] P. Li and B. Cui, "A comparative study on software vulnerability static analysis techniques and tools," in *2010 IEEE international conference on information theory and information security*. IEEE, 2010, pp. 521–524.

[3] A. Kaur and R. Nayyar, "A comparative study of static code analysis tools for vulnerability detection in c/c++ and java source code," *Procedia Computer Science*, vol. 171, pp. 2023–2029, 2020.

[4] Z. Shen and S. Chen, "A survey of automatic software vulnerability detection, program repair, and defect prediction techniques," *Security and Communication Networks*, vol. 2020, pp. 1–16, 2020.

[5] N. Ziems and S. Wu, "Security vulnerability detection using deep learning natural language processing," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2021, pp. 1–6.

[6] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, "Codebert: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 1536–1547.

[7] Y. Wang, W. Wang, S. Joty, and S. C. 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*, 2021, pp. 8696–8708.

[8] H. Hanif and S. Maffeis, "Vulberta: Simplified source code pre-training for vulnerability detection," in *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2022, pp. 1–8.

[9] 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," *arXiv preprint arXiv:1907.11692*, 2019.

[10] 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.

[11] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.

[12] 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 Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[13] Z. Liu, X. Yu, Y. Fang, and X. Zhang, "Graphprompt: Unifying pre-training and downstream tasks for graph neural networks," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 417–428.

[14] 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*, 2022, pp. 8410–8423.

[15] 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.

[16] 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*, 2021, pp. 255–269.

[17] 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*, 2021, pp. 4582–4597.

[18] R. Mao, Q. Liu, K. He, W. Li, and E. Cambria, "The biases of pre-trained language models: An empirical study on prompt-based sentiment analysis and emotion detection," *IEEE Transactions on Affective Computing*, 2022.

[19] G. Yang, Y. Zhou, X. Chen, X. Zhang, T. Han, and T. Chen, "Exploitgen: Template-augmented exploit code generation based on codebert," *Journal of Systems and Software*, vol. 197, p. 111577, 2023.

[20] K. Liu, G. Yang, X. Chen, and Y. Zhou, "El-codebert: Better exploiting codebert to support source code-related classification tasks," in *Proceedings of the 13th Asia-Pacific Symposium on Internetware*, 2022, pp. 147–155.

[21] C. Yu, G. Yang, X. Chen, K. Liu, and Y. Zhou, "Bashexplainer: Retrieval-augmented bash code comment generation based on fine-tuned codebert," in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2022, pp. 82–93.

[22] L. Buratti, S. Pujar, M. Bornea, S. McCarley, Y. Zheng, G. Rossiello, A. Morari, J. Laredo, V. Thost, Y. Zhuang *et al.*, "Exploring software naturalness through neural language models," *arXiv preprint arXiv:2006.12641*, 2020.

[23] H. Hanif, M. H. N. M. Nasir, M. F. Ab Razak, A. Firdaus, and N. B. Anuar, "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches," *Journal of Network and Computer Applications*, vol. 179, p. 103009, 2021.

[24] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang *et al.*, "Codexglue: A machine learning benchmark dataset for code understanding and generation," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[25] T. Gao, A. Fisch, and D. Chen, "Making pre-trained language models better few-shot learners," *arXiv preprint arXiv:2012.15723*, 2020.

[26] T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh, "Autoprompt: Eliciting knowledge from language models with automatically generated prompts," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 4222–4235.

[27] 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*, 2021, pp. 3045–3059.

[28] G. Lin, S. Wen, Q.-L. Han, J. Zhang, and Y. Xiang, "Software vulnerability detection using deep neural networks: a survey," *Proceedings of the IEEE*, vol. 108, no. 10, pp. 1825–1848, 2020.

[29] D. Zou, S. Wang, S. Xu, Z. Li, and H. Jin, "muvuldeepecker: A deep learning-based system for multiclass vulnerability detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, pp. 2224–2236, 2019.

[30] W. Fu and T. Menzies, "Easy over hard: A case study on deep learning," in *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, 2017, pp. 49–60.

[31] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris, "Spottune: transfer learning through adaptive fine-tuning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4805–4814.

[32] 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.

[33] 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.

[34] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet," *IEEE Transactions on Software Engineering*, 2021.

[35] 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.

[36] X. Wen, Y. Chen, C. Gao, H. Zhang, J. M. Zhang, and Q. Liao, "Vulnerability detection with graph simplification and enhanced graph representation learning," *arXiv preprint arXiv:2302.04675*, 2023.

[37] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[38] P. Budzianowski and I. Vulic, "Hello, its gpt-2-how can i help you? towards the use of pretrained language models for task-oriented dialogue systems," *EMNLP-IJCNLP 2019*, p. 15, 2019.

[39] 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: System Demonstrations*, 2022, pp. 105–113.

[40] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2018.

[41] J. Li, T. Tang, W. X. Zhao, J.-Y. Nie, and J.-R. Wen, "Pretrained language models for text generation: A survey," *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, 2022.

[42] T. H. Le, H. Chen, and M. A. Babar, "A survey on data-driven software vulnerability assessment and prioritization," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–39, 2022.

[43] S. Cao, X. Sun, L. Bo, R. Wu, B. Li, and C. Tao, "Mvd: memory-related vulnerability detection based on flow-sensitive graph neural networks," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1456–1468.

[44] Y. Shin and L. Williams, "An empirical model to predict security vulnerabilities using code complexity metrics," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, 2008, pp. 315–317.

[45] H. K. Dam, T. Tran, T. Pham, S. W. Ng, J. Grundy, and A. Ghose, "Automatic feature learning for vulnerability prediction," *arXiv preprint arXiv:1708.02368*, 2017.

[46] Z. Jiang, F. F. Xu, J. Araki, and G. Neubig, "How can we know what language models know?" *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 423–438, 2020.

[47] W. Yuan, G. Neubig, and P. Liu, "Bartscore: Evaluating generated text as text generation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 27 263–27 277, 2021.

[48] E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh, "Universal adversarial triggers for attacking and analyzing nlp," 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.

[49] 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*, 2022, pp. 382–394.

[50] X. Li, X. Ren, Y. Xue, Z. Xing, and J. Sun, "Prediction of vulnerability characteristics based on vulnerability description and prompt learning," in *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2023, pp. 604–615.