SCPatcher: Mining Crowd Security Discussions to Enrich Secure Coding Practices

Ziyou Jiang[1,2,3,†], Lin Shi[4,†], Guowei Yang[5], Qing Wang[1,2,3,*],
[1]State Key Laboratory of Intelligent Game, Beijing, China;
[2]Science and Technology on Integrated Information System Laboratory,
Institute of Software Chinese Academy of Sciences, Beijing, China;
[3]University of Chinese Academy of Sciences, Beijing, China;
[4]School of Software, Beihang University, Beijing, China;
[5]The University of Queensland, Brisbane, Australia
[†]Both authors contributed equally to this research, [*]Corresponding author;
Email: {ziyou2019, wq}@iscas.ac.cn, shilin@buaa.edu.cn, guowei.yang@uq.edu.au

***Abstract*—Secure coding practices (SCPs) have been proposed to guide software developers to write code securely to prevent potential security vulnerabilities. Yet, they are typically one-sentence principles without detailed specifications, e.g., "Properly free allocated memory upon the completion of functions and at all exit points.", which makes them difficult to follow in practice, especially for software developers who are not yet experienced in secure programming. To address this problem, this paper proposes SCPatcher, an automated approach to enrich secure coding practices by mining crowd security discussions on online knowledge-sharing platforms, such as Stack Overflow. In particular, for each security post, SCPatcher first extracts the area of coding examples and coding explanations with a fix-prompt tuned Large Language Model (LLM) via Prompt Learning. Then, it hierarchically slices the lengthy code into coding examples and summarizes the coding explanations with the areas. Finally, SCPatcher matches the CWE and Public SCP, integrating them with extracted coding examples and explanations to form the SCP specifications, which are the wild SCPs with details, proposed by the developers. To evaluate the performance of SCPatcher, we conduct experiments on 3,907 security posts from Stack Overflow. The experimental results show that SCPatcher outperforms all baselines in extracting the coding examples with 2.73% MLine on average, as well as coding explanations with 3.97% F1 on average. Moreover, we apply SCPatcher on 447 new security posts to further evaluate its practicality, and the extracted SCP specifications enrich the public SCPs with 3,074 lines of code and 1,967 sentences.**

I. Introduction

Secure Coding Practices (SCPs) have been recently proposed by practitioners and institutions to guide software developers to write code that is more resistant to security vulnerabilities and attacks [1]. For example, Google proposed "*Best Practices for Security & Privacy*" for secure development on Android [2]; University of California, Berkeley proposed "*Secure Coding Practice Guidelines*" to guide developers regarding the application software security [3]. The Open Worldwide Application Security Project (OWASP) proposed a quick reference for security in general coding [4], which is a comprehensive set of SCPs.

However, the aforementioned publicly available SCPs are typically one-sentence principles without detailed specifications. For example, there is an SCP from OWASP regarding memory management: "*Properly free allocated memory upon the completion of functions and at all exit points*", which is barely a single sentence recommending developers to "*properly free allocated memory*" without any detailed guidance. Thus, such SCPs are difficult to follow in practice, especially for software developers who are not yet experienced in secure programming. On the other hand, there are many crowd security discussions on online knowledge-sharing platforms, such as Stack Overflow (SO) [5], and some security posts incorporate detailed coding practices, including **coding examples** and **explanations**, which are even referenced by security practitioners. For example, Frank van Puffelen suggested the best practices for Google Firebase data models on Twitter [6] with the SO security post #70711696. Therefore, the detailed coding practices embedded in the crowd security discussions are complementary to the public SCPs and can be leveraged to form the SCP specifications to enrich these SCPs.

In this paper, we propose SCPatcher, an automated approach to enrich secure coding practices by mining crowd security discussions on online knowledge-sharing platforms. In particular, for each security post, SCPatcher first extracts the area of coding examples and explanations with the LLM, which is the novel method that achieves state-of-the-art (SOTA) performances on multiple natural language process (NLP) tasks. We guide the area extraction with Prompt Learning on LLM and utilize the fix-prompt tuning to train it on our dataset. Then, it hierarchically slices the lengthy code to the coding examples and summarizes the coding explanations within these areas, Finally, SCPatcher matches the CWE and public SCP, and integrates them with extracted coding examples and explanations to form the SCP specifications.

To evaluate the performance of SCPatcher, we conduct experiments on 3,907 security posts collected from Stack Overflow, and compare SCPatcher with multiple representative baselines. The results show that, SCPatcher outperforms all baselines on extracting the coding examples, outperforming the baselines with 2.73% on the matching rate of LOC (MLine) on average. SCPatcher also achieves the highest

F1 performances on extracting coding explanations, outperforming the baselines with 3.97% on average. For enriching the CWE and public SCP, we apply SCPatcher on 447 new security posts to further evaluate the practical usage. The extracted SCP specifications enrich the public SCP with 3,074 LOC and 1,967 sentences. The major contributions of this paper are summarized as follows:

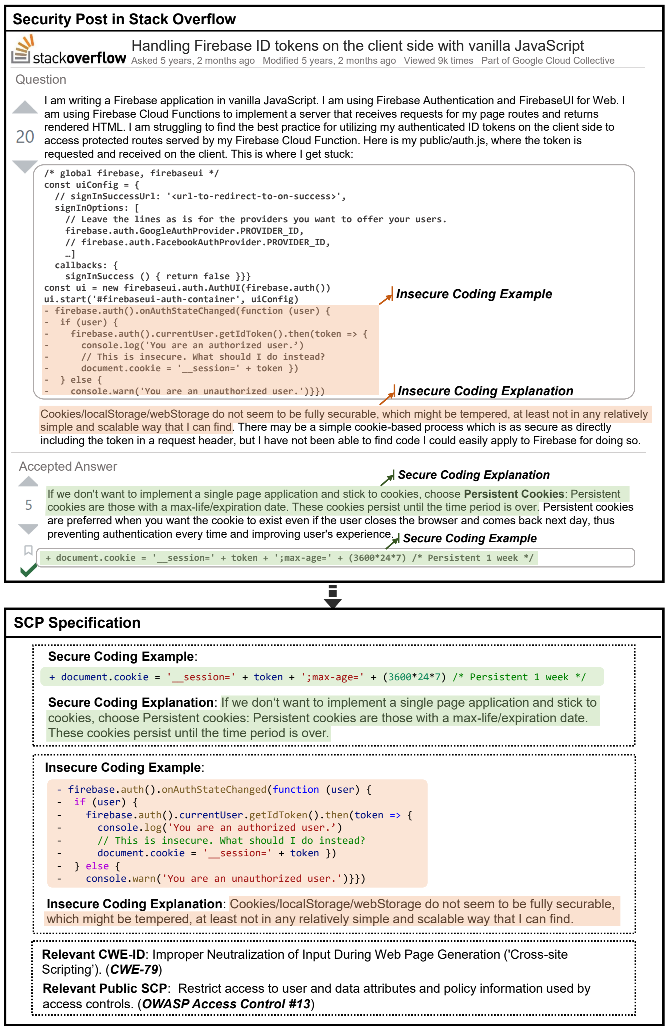
- **Technique**: SCPatcher, an automated approach to enrich the public secure coding practices. To the best of our knowledge, this is the first work on automatically enriching the public SCPs with crowd security discussions.
- **Evaluation**: An experimental evaluation of SCPatcher, which shows that SCPatcher outperforms all baselines, together with a user study with security practitioners, which further demonstrates its usefulness in practice.
- **Data**: We release a public dataset with 3,907 security posts and source code on [7] to facilitate the replication and the application of SCPatcher in the more extensive contexts.

In the rest of the paper, Section II illustrates the motivation example. Section III presents the details of our approach. Section IV sets up the experiments. Section V describes the experimental results and analysis. Section VI presents the discussion and threats to validity. Section VII discusses the related work, and Section VIII concludes this paper.

II. Motivating Example

Recent researchers have observed that open-source community platforms such as Stack Overflow and GitHub have converged security-related knowledge in a large volume [8]. Although the crowd and open nature might lower the confidence in the accuracy of those security reports released by organizations, the platforms still have non-negligible contributions in providing valuable and practical knowledge to help massive developers resolve their security concerns. Mining the crowd security discussions on those platforms would give the opportunity to enrich the current secure coding practices.

Fig. 1 shows an example of enriched secure coding practices from Stack Overflow's security posts. One developer asked a security question about authenticating the Firebase using cookies, providing the insecure code and the corresponding explanation that the cookies might be tempered. The post has been viewed 9K times, and the accepted answer contains the detailed secure practice of using the *persistent cookies* to avoid repeated login in the Firebase. From their discussions, we can extract both secure and insecure coding examples and their corresponding explanations as shown in Fig. 1. Moreover, we find that their discussed insecure coding example and explanation indicate the tempering of cookies from the web pages, which could match with the "cross-site scripting" weakness reported in CWE-79 [9]. The secure coding example and explanation introduce the protection of such weakness by restricting access to users, which matches with the Access Control #13 reported by OWASP [4]. Based on the observation, we believe that extracting detailed examples and their explanations, as well as matching them with current secure



Handling Firebase ID tokens on the client side with vanilla JavaScript
Asked 5 years, 2 months ago Modified 5 years, 2 months ago Viewed 9k times Part of Google Cloud Collective

I am writing a Firebase application in vanilla JavaScript. I am using Firebase Authentication and FirebaseUI for Web. I am using Firebase Cloud Functions to implement a server that receives requests for my page routes and returns rendered HTML. I am struggling to find the best practice for utilizing my authenticated ID tokens on the client side to access protected routes served by my Firebase Cloud Function. Here is my public/auth.js, where the token is requested and received on the client. This is where I get stuck:

```
/* global firebase, firebaseui */
const uiConfig = {
  // signInSuccessUrl: '<url-to-redirect-to-on-success>',
  signInOptions: [
    // Leave the lines as is for the providers you want to offer your users.
    firebase.auth.GoogleAuthProvider.PROVIDER_ID,
    // firebase.auth.FacebookAuthProvider.PROVIDER_ID,
    …]
  callbacks: {
    signInSuccess () { return false }}}
const ui = new firebaseui.auth.AuthUI(firebase.auth())
ui.start('#firebaseui-auth-container', uiConfig)
- firebase.auth().onAuthStateChanged(function (user) {
-  if (user) {
-    firebase.auth().currentUser.getIdToken().then(token => {
-      console.log('You are an authorized user.’)
-      // This is insecure. What should I do instead?
-      document.cookie = '__session=' + token })
-  } else {
-    console.warn('You are an unauthorized user.')}})
```

Cookies/localStorage/webStorage do not seem to be fully securable, which might be tempered, at least not in any relatively simple and scalable way that I can find. There may be a simple cookie-based process which is as secure as directly including the token in a request header, but I have not been able to find code I could easily apply to Firebase for doing so.

If we don't want to implement a single page application and stick to cookies, choose **Persistent Cookies**: Persistent cookies are those with a max-life/expiration date. These cookies persist until the time period is over. Persistent cookies are preferred when you want the cookie to exist even if the user closes the browser and comes back next day, thus preventing authentication every time and improving user's experience.

```
+ document.cookie = '__session=' + token + ';max-age=' + (3600*24*7) /* Persistent 1 week */
```

Secure Coding Example:

```
+ document.cookie = '__session=' + token + ';max-age=' + (3600*24*7) /* Persistent 1 week */
```

Secure Coding Explanation: If we don‘t want to implement a single page application and stick to cookies, choose Persistent cookies: Persistent cookies are those with a max-life/expiration date. These cookies persist until the time period is over.

Insecure Coding Example:

```
- firebase.auth().onAuthStateChanged(function (user) {
-  if (user) {
-    firebase.auth().currentUser.getIdToken().then(token => {
-      console.log('You are an authorized user.’)
-      // This is insecure. What should I do instead?
-      document.cookie = '__session=' + token })
-  } else {
-    console.warn('You are an unauthorized user.')}})
```

Insecure Coding Explanation: Cookies/localStorage/webStorage do not seem to be fully securable, which might be tempered, at least not in any relatively simple and scalable way that I can find.

Relevant CWE-ID: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'). (***CWE-79***)

Relevant Public SCP: Restrict access to user and data attributes and policy information used by access controls. (***OWASP Access Control #13***)

Fig. 1: An example of extracting SCP specification from Stack Overflow #48884217.

coding practices, could enable a better understanding and thus facilitate the development of secure software.

III. Approach

The overall framework of SCPatcher is illustrated in Fig. 2. Since directly extracting the coding examples and explanations from the entire security post is non-trivial, we adopt a two-step extraction strategy: first, we extract the areas of coding examples and explanations with LLM; second, from the extracted areas, we condense the coding examples with hierarchically slicing if they are lengthy and extract the sentences about coding explanations. Afterwards, we match the extracted coding examples and explanations to the relevant CWE and public SCP and put them together to form an SCP specification.

A. Extracting the Areas of Coding Example and Explanation

A security post typically contains many sentences, while only a relatively small number of sentences are relevant to coding examples and explanations. To reduce the impact of

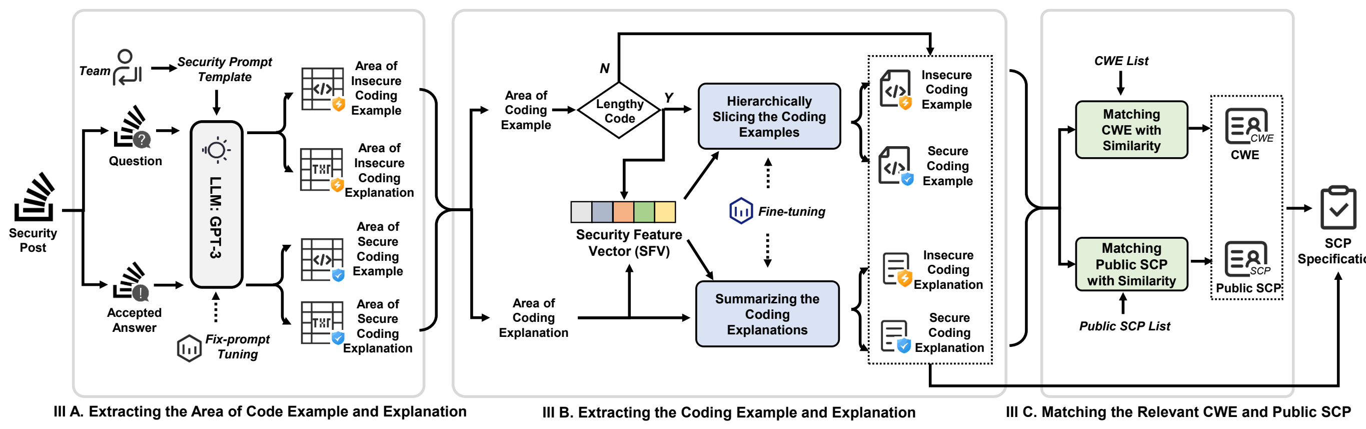

Team
Security Prompt Template
Question
Accepted Answer
Security Post
LLM: GPT-3
Fix-prompt Tuning
Area of Insecure Coding Example
Area of Insecure Coding Explanation
Area of Secure Coding Example
Area of Secure Coding Explanation
Area of Coding Example
Lengthy Code
N
Y
Hierarchically Slicing the Coding Examples
Security Feature Vector (SFV)
Fine-tuning
Area of Coding Explanation
Summarizing the Coding Explanations
Insecure Coding Example
Secure Coding Example
Insecure Coding Explanation
Secure Coding Explanation
CWE List
Matching CWE with Similarity
CWE
Matching Public SCP with Similarity
Public SCP
Public SCP List
SCP Specification
III A. Extracting the Area of Code Example and Explanation
III B. Extracting the Coding Example and Explanation
III C. Matching the Relevant CWE and Public SCP

Fig. 2: The overview of our approach.

non-relevant sentences, we extract the areas that are more likely to contain examples and explanations.

Typically, developers tend to propose security concerns in Questions, and other developers would give solutions in Accepted Answers. Therefore, Given the security post $\mathcal{P}$, we predict the areas of *insecure* coding examples and explanations from question Q, and predict the *secure* coding examples and explanations from accepted answer A.

To better fit the LLM to our task, we use the Prompt Learning [10] technique for tuning the LLM. Prompt Learning is a novel paradigm for the prediction and training of LLM, achieving SOTA performance on various NLP tasks. Different from the traditional paradigm, i.e., Pre-training+Fine-tuning, prompt learning adapts the NLP tasks to the LLMs [11], which reduces the training costs and fully utilizes the resources of models. Prompt learning utilizes the *Prompt Templates* [12] to enhance inputs with instructions of NLP tasks and bridge the gaps between inputs and LLMs.

Prompt-based Area Extraction. To extract the areas based on the prompt-learning, we design the prompt template based on the **Cloze-testing**, which is a typical prompt template in text extraction tasks [13]. We compare the two widely-used templates, i.e., **Cloze-testing** and **Prefix** templates, and choose the best-performed one (Section VI-A). The template for extracting the areas is shown as follows:

- ***Prompt Template***: *[X] The* $\{insecure|secure\}$ *codes and sentences are [Y].*

where the [X] indicates the input of LLM, i.e., Q and A. We first replace the [X] token in the template with Q and A, and obtain the templated inputs T_Q and T_A:

- $\boldsymbol{T_Q}$: *[Q]. The insecure codes and sentences are [Y].*
- $\boldsymbol{T_A}$: *[A]. The secure codes and sentences are [Y].*

where [Q] and [A] indicate the texts in question and the accepted answer. Second, we choose the generative LLM to predict the [Y] token in the T_Q and T_A [14]. Respectively, the output [Y] of T_Q consists of the area of insecure coding example A_Exam^{-}, and the area of insecure coding explanation A_Expl^{-}; the output [Y] of T_A consists of the area of secure coding example A_Exam^{+} and the area of secure coding explanation A_Expl^{+}.

Fix-prompt Tuning. For the training of LLM, we apply the Fix-prompt Tuning [15] which is a training method for manually designed templates, and is more suitable for the few-shot scenario. Given the predicted areas and ground-truth labels, the loss for fix-prompt tuning is calculated as follows:

$$\begin{aligned}\mathcal{L}_{area} = \lambda_1 Cr(y_{exam^-}, A\_Exam^-) + \lambda_2 Cr(y_{exam^+}, A\_Exam^+) \\ + \lambda_3 Cr(y_{expl^-}, A\_Expl^-) + \lambda_4 Cr(y_{expl^+}, A\_Expl^+)\end{aligned} \quad (1)$$

where y_{exam^-}, y_{exam^+}, y_{expl^-}, and y_{exam^+} indicate the labeled insecure&secure coding examples, and insecure&secure coding explanations. The function $Cr(y, f(x))$ is the CRINGE loss [16], which is specifically used to train generative LLM models. $\lambda_1 \sim \lambda_4$ are the loss-balancing parameters for the fix-prompt tuning.

To select the best LLM for extracting the areas, we choose five representative LLMs, i.e., BERT-base [17], Albert-large [18], GPT-2 [19], T5 [20], and GPT-3 [21], [22], and conduct the fix-prompt tuning on our labeled dataset in Section IV-A, We compare the accuracy between predicted and ground-truth areas, which calculates the ratio of areas that contain coding examples and explanations on all the predictions. We can see that, as is shown in Table I, the GPT-3 achieves the highest results on the four tasks of area extraction, with over 80% accuracy. It outperforms the other LLMs with 6.48% (Insecure Coding Example), 2.77% (Secure Coding Example), 8.50% (Insecure Coding Explanation), and 4.27% (Secure Coding Explanation). Therefore, we choose GPT-3 as the LLM in SCPatcher.

TABLE I: The accuracy of LLM on extracting the areas of coding examples and explanations (%).

LLM	Coding Example		Coding Explanation	
	Insecure	Secure	Insecure	Secure
BERT-base	65.76	65.35	68.53	59.92
Albert-large	70.92	68.59	73.45	63.07
GPT-2	70.44	70.23	73.92	73.14
T5	82.58	84.83	78.22	79.36
GPT-3	**89.06 (↑6.48)**	**87.60 (↑2.77)**	**86.72 (↑8.50)**	**83.65 (↑4.27)**

TABLE II: The five categories of SFV's keywords.

Types	Description	Example
WW	The weakness words in the secure and insecure coding explanation.	Weakness Encoding, Compiler Optimization, Improper Handling,
TW	The target words in the secure and insecure coding examples.	Password, OS System, Code, Cookies
AW	The attack words in insecure coding explanations.	Change, Modify, Inject, Steal, Hack
DW	The defense words in secure coding explanations.	Prevent, Protect, Save, Initialize
CW	The words that both occur in insecure&secure coding explanations.	Memory, Method, Legitimate

B. Extracting Coding Examples and Explanations

The areas extracted by LLM contain insecure or secure coding examples and their explanations. To exactly extract these elements from the areas, we first extract the **security feature vector** (SFV) from the areas to represent the high-level security-related features. Then, based on the extracted areas and SFV, we slice the lengthy coding examples, and summarize the sentences of coding explanation in parallel. Finally, we output the insecure/secure coding examples and explanations.

Extracting the Secure Feature Vector (SFV). To accurately extract the security-related codes and sentences from the areas, we extract the SFV from the areas of coding examples and explanations, which are high-level representations of security-related knowledge. As shown in Table II, we extend Shen's work [23] and come up with five categories for SFV's keywords, i.e., Target Words (TW), Attack Words (AW), Defense Words (DW), and newly add Weakness Words (WW) and Co-occurrence Words (CW) for our task.

After extracting the words of in each category, we embed these words using GloVe [24], which is a widely-used word embedding model in NLP tasks, such as text classification, text summarization, semantic analysis, etc. The SFV is formed by concatenating all the word embeddings as follows:

$$\boldsymbol{\xi} = concat[GloVe(WW, TW, AW, DW, CW)] \quad (2)$$

where $GloVe$ is the GloVe embedding, $concat$ is the concatenation function, $WW, ..., CW$ represent the words in each category, and $\boldsymbol{\xi}$ is the extracted SFV.

Attention-based Selector. The Attention-based Selector (AttnSelector) is the core module for slicing lengthy codes and summarizing the coding explanations. As is shown in Fig. 3, the selector first utilize the **Task-oriented Encoder** to embed the elements of each area $[E_1, E_2, ..., E_n]$ to $[\boldsymbol{e}_1, \boldsymbol{e}_2, ..., \boldsymbol{e}_n]$. Then, it incorporates the SFV $\boldsymbol{\xi}$ to these embeddings with Multi-head Attention [25], which is an effective method to enhance the embeddings with external knowledge [26]. The processes of multi-head attention are shown as follows:

$$\begin{aligned} & f(\boldsymbol{e}_i, \boldsymbol{\xi}) = tanh(\boldsymbol{W}_c concat(\boldsymbol{e}_i, \boldsymbol{\xi}) + b) \\ & \alpha_i = softmax(f(\boldsymbol{e}_i, \boldsymbol{\xi})) \\ & \boldsymbol{c} = \sum_{i=1,...,n} \alpha_i \boldsymbol{e}_i \end{aligned} \quad (3)$$

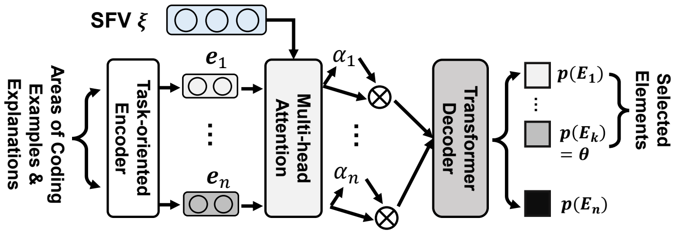


Fig. 3: The structure of Attention-based Selector.

where α_i indicates the attention weight of each element. $\boldsymbol{W}_c$ and b are the trainable parameters in attention-weight calculation. $softmax$ is the activation function to normalize the α_i. Then we sum the weighted embeddings as the enhanced embedding $\boldsymbol{c}$.

Finally, the selector utilizes the transformer decoder [27] to predict the **selection probabilities** that decide which element E_i should be selected as the output sentences or codes.

$$\begin{aligned} & [p(E_1), p(E_2), ..., p(E_n)] = Transformer(\boldsymbol{c}) \\ & Output = \{E_i | p(E_i) > \theta\} \end{aligned} \quad (4)$$

where $p(E_i)$ indicates the output probability of each element E_i. Based on these probabilities, we set the cut-off value θ and select elements with probabilities higher than θ as the summarized coding examples and explanations. For the two tasks in SCPatcher, the θ value is represented as θ_{slc} in extracting coding examples, and θ_{sum} in extracting coding explanations. We utilize the greedy search to set the θ that achieves the highest performance on each task.

Determination on Lengthy Code. Previous studies indicate that the developers tend to post their complete source code to the security post's code blocks, wrapped by <code> and </code> [28], [29]. These codes are usually lengthy, making it difficult for readers to understand their meanings. Some researchers suggest that one code block may not contain codes with more than 15 LOC [30]. To determine the cut-off value max_len for the lengthy code, we also manually inspect 100 code examples in CWEs, and confirm that all of their example codes are less than 15 LOC. Therefore, we determine to take the 15 as the threshold for lengthy code.

Hierarchically Slicing Coding Examples. Under the constraint of the maximum code length, we slice the insecure or secure code in terms of `function` and `fragment` based on the code areas. The slicing algorithm is shown in Algorithm 1. First, we initialize the *AttnSelector*'s task-oriented encoder with a novel encoder **CAST encoder** [31] that embeds the AST to the vector and initialize the SLC_Code with LEN_Code. Second, for the lengthy code, we transfer them to the Abstract Syntax Tree (AST) [32] by using the AST parser tools [33]. Third, based on the parsed nodes in AST, we separately split the AST based on the `function` (*func* nodes) and `fragment` (*comment* and *empty_line* nodes). Finally, we utilize the *AttnSelector* to select the candidate sub-trees with the slicing threshold θ_{slc}, and concatenate the selected sub-trees to the new SLC_Code. We iteratively conduct the split&select processes until the LOC is less than

Algorithm 1: Process of Hierarchical Code Slicer

```
Input: Original Lengthy Code $LEN\_Code$; SFV $\boldsymbol{\xi}$; Cut-off
       Value $\theta_{slc}$.
Output: Sliced Code $SLC\_Code$.
$Task\_Encoder \leftarrow CAST\_Encoder$;
$SLC\_Code \leftarrow LEN\_Code$;
while $len(SLC\_Code) > max\_len$ (15) do
    $AST = ASTParser(SLC\_Code)$;
    if $func \in AST$ then
        $[FC_1, ..., FC_n] = SplitFunc(AST)$;
        $SLC\_Code =$
          $concat(AttnSelector(FC_1, ..., FC_n, \boldsymbol{\xi}, \theta_{slc}))$;
    end
    if $comment/empty\_line \in AST$ then
        $[FG_1, ..., FG_n] = SplitFrag(AST)$;
        $SLC\_Code =$
          $concat(AttnSelector(FG_1, ..., FG_n, \boldsymbol{\xi}, \theta_{slc}))$;
    end
end
return $SLC\_Code$;
```

max_len, and output the SLC_Code. We input the area of insecure coding example A_Exam^- to the slicer and obtain the output insecure coding example $Exam^-$, and input the area of secure coding example A_Exam^+ to obtain the secure coding example $Exam^+$.

Summarizing Coding Explanations. In this step, we aim to summarize secure/insecure coding explanations from the extracted areas. Given the area of secure/insecure coding explanations A_Expl, which consists of a set of sentences $[S_1, S_2, ..., S_n]$. We choose the **BERT encoder** [17] as the Task-oriented Encoder to embed these sentences, which is a bidirectional Masked Language Model and Next Sentence Prediction, and has achieved SOTA performances in many NLP tasks [34], [35]. Finally, we utilize the $AttnSelector$ to summarize the sentences of coding explanations with the cut-off value θ_{sum}:

$$Expl = AttnSelector(S_1, ..., S_n, \boldsymbol{\xi}, \theta_{sum}) \tag{5}$$

where $Expl$ is the summarized coding explanations. We summarize the insecure coding explanation $Expl^-$ from the Question explanation area, and summarize the secure coding explanation $Expl^+$ from the Answer explanation area.

Fine-tuning. Given the selection probabilities in slicing coding examples and summarizing the coding explanations, we calculate the combined loss as follows:

$$\begin{aligned}\mathcal{L} = \lambda_{slc} \sum_{k=1}^{K} [H(y_{exam}, p(FC)) + H(y_{exam}, p(FG))] \\ + H(y_{expl}, p(S))\end{aligned} \tag{6}$$

where $p(FC)$ indicates the probabilities of `function`-based code slicer; $p(FG)$ indicates the probabilities in `fragment`-based code slicer; $p(S)$ indicates the probabilities in the coding explanation summarizer. y_{exam} and y_{expl} indicate the ground-truth of coding examples and explanations. $H(y, p(x))$ is the cross-entropy loss function. K is the number of iterations in slicing the coding examples, and λ_k is the loss-balancing parameter for each iteration k. We fine-tune the hierarchical slicer of coding examples, and the summarizer of coding explanations, with the joint loss $\mathcal{L}$, until both models achieve convergence.

C. Matching the Relevant CWE and Public SCP

This step aims to match the extracted coding examples and explanations to the relevant CWE and public SCP based on **security similarity**. Given the extracted insecure coding examples and explanations $C^- = \{Exam^-, Expl^-\}$, and secure coding examples and explanations $C^+ = \{Exam^+, Expl^+\}$, we first obtain the SFV keywords in C^- and C^+. For each SFV keyword w_i, we find the most similar words in the CWE or public SCP based on *cosine similarity*. Finally, we calculate the average value of all the SFV keywords.

$$\begin{aligned} sim_{cwe} = Avg_{(\boldsymbol{w}_i \in C^-)} max_{(\boldsymbol{w}_j \in CWE)} cos(\boldsymbol{w}_i, \boldsymbol{w}_j) \\ sim_{scp} = Avg_{(\boldsymbol{w}_i \in C^+)} max_{(\boldsymbol{w}_j \in SCP)} cos(\boldsymbol{w}_i, \boldsymbol{w}_j)\end{aligned} \tag{7}$$

where $\boldsymbol{w}_i$ indicates the embeddings of SFV keywords, and $\boldsymbol{w}_j$ indicates the words in CWE/public SCP. The function cos indicates the cosine similarity, and function max calculates the distance between the SFV keywords and their most similar word in the CWE or public SCP. For each security post, we select the CWE and public SCP when the *maximum similarity* is higher than the thresholds θ_{cwe} and θ_{scp}. For the similarity lower than the thresholds, we consider the CWE or public SCP as "unmatched", which may contain novel weaknesses and SCPs that are not currently included in CWE and public SCP. We form the output SCP specification by combining the selected CWE, public SCP, with the secure/insecure coding examples and explanations.

IV. Experimental Design

To evaluate the performance of SCPatcher, we investigate the following three research questions:

- **RQ1**: What are the performances of SCPatcher on extracting the secure or insecure coding example?
- **RQ2**: What are the performances of SCPatcher on extracting the secure or insecure coding explanation?
- **RQ3**: What are the performances and capability of SCPatcher on matching the CWE?
- **RQ4**: What are the performances and capability of SCPatcher on matching the public SCP?

A. Dataset Preparation

In this section, we prepare our dataset in four steps, i.e., data collection and filtering, data preprocessing, data labeling, and data augmentation.

Data Collection and Filtering Following the previous study [36], we collect security posts from Stack Overflow via Stack Exchange Data Explorer [37]. Specifically, we retrieve security posts from the beginning until Aug 31, 2022, and prepare the dataset in the following procedures: 1) We obtain all the posts that are tagged with "**security**" and its similar tags, such as "websecurity", "danger" and "firebase", etc., according to Yang's work [38]. 2) We filter out the security

TABLE III: The number of security posts in our dataset.

Items	Dataset	#Total	#SCP	#Sentences	#LOC
Origin	*Train*	3,126	1,874	20,375	19,056
	Test	781	447	15,247	10,036
Augmented	*Train*	10,314	9,772	87,204	80,134
	Test	781	447	15,247	10,036

posts that do not contain <code> HTML tags, and retain the posts that tend to discuss the secure coding practice. We manually inspect the security posts and find that, among those posts that do not contain <code> HTML tags, less than 3% posts contain the content of SCP specifications, thus removing them has a small impact on our dataset. 3) We filter out the posts that receive negative scores voted by Stack Overflow users, as well as non-English posts. 4) We filter out the posts that the scores of their **accepted answers** are negative to avoid the bias of subjective decision on accepted answers. As is shown in **Origin** row of Table III, we collect 3,907 posts, with 2,321 SCP specifications, 24.589 sentences, and 22,761 LOC in our original dataset.

Data Preprocessing. We preprocess the collected security posts with the following procedures: 1) We use the pipeline to correct typos, remove stopwords, and lemmatize the texts with Spacy by following the previous work [8]; 2) We tag the code blocks (wrapped by HTML tags <code>, </code>) with the token `[CODE]`; and 3) We remove other HTML tags, such as <p>, <li>, etc., and retain the plain text inside.

Data Labeling. For each security post, we label the codes of insecure and secure coding examples, as well as the sentences of insecure and secure coding explanations. We build a team with two senior researchers, two Ph.D. students, and four Master's students to label the dataset. All annotators have extensive experience in secure software development, and four participants (50%) are external annotators. To reduce the bias, each security post is labeled by two team members. When different opinions occur on the labels, we discuss them with all team members until a decision has been reached. Only a few posts have conflicting opinions, and the average Cohen's Kappa [39] is 0.9[1], which means the biases of ground-truth labeling are minor.

Data Augmentation. As introduced in section III-A, we perform prompt learning to tune LLM, and fine-tune the $AttnSlicer$ to fit our task. We split the dataset into training and testing datasets with the proportions 80% and 20%, where the training dataset is used for tuning LLM and $AttnSlicer$, and the testing dataset is used for evaluation. To make the tuning sufficient and effective, we augment the training dataset by employing EDA [40], a widely-used and effective data augmentation technique. By replacing words with synonyms, random insertion, deletion, or swapping of words, etc., EDA helps us create new security posts while keeping their original meanings. The details of the augmented dataset are shown in the **Augmented** row. In total, we collect 3,907 security posts and augment them to 11,091 security posts.

[1]The kappa-value>0.81 is considered as perfect agreement.

B. Baseline Selection

To evaluate the advantage of our approach, we select a SOTA approach on general NLP tasks (i.e., GPT-3), two SOTA approaches on vulnerable code slicing (i.e., VulSlicer and DeepBalance), and two SOTA approaches on text summarization (i.e., BERTSum and BART).

Baseline for Both RQ1 and RQ2. To compare the SC-Patcher with LLMs, we choose the **GPT-3** [21] as the common baselines on both RQ1 and RQ2, which is a novel LLM proposed by OpenAI. GPT-3 has achieved SOTA performances on many few-shot NLP tasks, and it also opens the fine-tuning interface for researchers to train their own models. We utilize the same prompt template in Section III-A to directly extract the coding examples and explanations, then use the labeled dataset to fine-tune the GPT-3.

Baselines for RQ1. VulSlicer [41] is a search-based program slicer that slices the vulnerable code by comparing the code embeddings with the codes in the pre-defined vulnerability library. **DeepBalance** [42] is the novel vulnerable code detector [43] that utilizes the Bidirectional LSTM to learn invariant and discriminative code representations and detect the vulnerable codes, and employs the fuzzy oversampling method to train the model.

Baselines for RQ2. BERTSum [44] is a novel text summarization model that fine-tunes BERT model with summarization-based tokenizer and use the embedded sentence to predict whether it belongs to the summarization; **BART** [45] is a seq-to-seq denoise auto-encoder trained on corrupting texts, and is suitable for information extraction of documents with various structures.

To ensure a fair comparison, we performed fine-tuning for all the baselines on our data with the same settings as our approach. Moreover, to make the baselines more comparable with our approach, we provide various advanced inputs for these baselines.

- **Baseline+Area**. For each baseline, we input the extracted areas of code and explanations into them, instead of the entire Stack Overflow posts.
- **Baseline+Area+SFV**. For each baseline, we input the areas combined with our SFV embeddings.

C. Evaluation Metric Selection

For RQ1, we apply the generative **Rouge-L** [46] to measure the sliced coding examples. We also chosen widely-used **matching rate** of **tokens** (**MToken**) [47] and **lines** (**MLine**) [41] to calculate the similarity between predicted and ground-truth codes.

To evaluate the RQ2, we apply three commonly-used metrics: 1) **Precision** (**Pre.**), which calculates the ratio of correct positive predictions to the total positive predictions; 2) **Recall** (**Rec.**), which calculates the ratio of correct positive predictions to the ground-truth positive labels; 3) **F1-measure** (**F1**), which is the harmony value of precision and recall.

TABLE IV: The baseline comparison on extracting the insecure and secure coding examples (%).

Variants	Models	Training Hours	Insecure Coding Examples			Secure Coding Examples			*Average*		
			Rouge-L	MToken	MLine	Rouge-L	MToken	MLine	Rouge-L	MToken	MLine
Baseline	VulSlicer	**8h**	44.36	40.69	38.02	48.90	58.09	56.04	46.63	49.39	47.03
	DeepBalance	7h	46.72	53.45	55.19	51.35	63.49	65.05	48.04	58.47	60.12
	GPT-3	>20h	54.50	56.25	52.13	56.92	65.25	66.42	55.71	60.75	59.28
Baseline+ Area	VulSlicer	11.5h	49.75	53.25	55.13	51.05	58.24	56.57	50.40	55.75	55.85
	DeepBalance	12h	54.12	59.24	63.17	57.83	66.85	67.02	55.98	63.05	65.10
	GPT-3	>20h	56.80	64.06	68.75	61.81	71.42	70.54	59.31	67.74	69.65
Baseline+ Area+SFV	VulSlicer	13h	51.13	57.72	59.60	58.95	62.28	63.19	55.04	60.00	61.40
	DeepBalance	15h	56.08	69.18	66.28	60.47	69.45	71.63	58.28	69.32	68.96
	GPT-3*	-		-			-			-	
Our Model	**SCPatcher**	11h (↑**3h**)	**60.03** (↑**3.23**)	**71.40** (↑**2.22**)	**73.31** (↑**4.56**)	**66.53** (↑**4.72**)	**77.06** (↑**5.64**)	**75.44** (↑**3.81**)	**63.33** (↑**4.03**)	**74.23** (↑**4.91**)	**72.38** (↑**2.73**)

TABLE V: The baseline comparison on extracting the insecure and secure coding explanations (%).

Variants	Models	Training Hours	Insecure Coding Explanation			Secure Coding Explanation			*Average*		
			Pre.	Rec.	F1	Pre.	Rec.	F1	Pre.	Rec.	F1
Baseline	BERTSum	**6h**	57.72	60.29	58.98	62.37	61.15	61.75	60.05	60.72	60.37
	BART	11h	60.24	64.32	62.21	66.42	67.02	66.72	63.33	65.67	64.47
	GPT-3	16h	66.15	67.67	66.90	69.15	68.06	68.60	67.65	67.87	67.75
Baseline +Area	BERTSum	17h	62.65	68.96	65.65	65.87	69.25	67.52	64.26	69.11	66.59
	BART	>20h	65.47	68.06	66.74	71.74	70.80	71.27	68.61	69.43	69.00
	GPT-3	>20h	74.53	75.94	75.23	76.25	75.64	75.94	75.39	75.79	75.59
Baseline +Area+SFV	BERTSum	18h	65.37	70.49	67.83	68.95	68.14	68.54	67.16	69.32	68.19
	BART	>20h	67.52	69.92	68.70	73.45	73.52	73.48	70.49	71.72	71.09
	GPT-3*	-		-			-			-	
Our Model	**SCPatcher**	11h (↑**5h**)	**78.24** (↑**3.71**)	**77.92** (↑**1.98**)	**78.08** (↑**2.85**)	**80.62** (↑**4.37**)	**81.47** (↑**5.83**)	**81.04** (↑**5.10**)	**79.43** (↑**4.04**)	**79.70** (↑**3.91**)	**79.56** (↑**3.97**)

*Due to the temporary lack of open embeddings for GPT-3, we will compare the GPT-3+Area+SFV when the embeddings are available.

To evaluate the RQ3 and RQ4, we utilize the number of LOC and sentences to measure the enrichment of CWE and public SCP, and apply the **Precision** to measure the performances of the matchers, which can reflect the accurate proportion of the matched CWE and public SCP.

D. Experiment Settings

To answer RQ1 and RQ2, for the SCPatcher, we first set the max_len as 15. Then, based on the grid searching [48], we set the thresholds θ_{slc}, θ_{sum} as 0.6. We set the loss-balancing parameters $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 0.25$, $\lambda_{slc} = 0.5$ by grid search, and choose the 16 *batch_size* to train the SCPatcher according to the hardware limitations.

For all the baselines and their variants, we choose the default parameter settings in these works, and use the same *batch_size* 16 to train the models. All the training processes are run on a personal computer with Windows 11 OS, NVIDIA GeForce RTX 2060 GPU, and 32GB RAM.

To answer RQ3 and RQ4, we set the θ_{cwe}, θ_{scp} as 0.5 based on the greedy search. We analyze the performances of matching the CWE and public SCP, where we use the OWASP guidelines as the public SCP, since it is a relatively comprehensive public SCP on various general coding practices.

V. Results

A. Performance on Extracting the Coding Examples

Table IV illustrates the comparison results of SCPatcher on matching the coding examples, and the best performance of each column is highlighted with **bold face**.

Comparison with Original Baselines. Comparing the SCPatcher with the original baselines, we can see that, SCPatcher outperforms all the original baselines on average, improving the highest baseline by 7.62% (Rouge-L), 13.48% (MToken), and 12.26% (MLine).

Comparison with Enhanced Baselines. Comparing the SCPatcher with the enhanced baselines, i.e., Baseline+Area and Baseline+Area+SFV, we can see that, SCPatcher also outperforms these enhanced baselines on average, improving the best baseline GPT3+Area by 4.03% (Rouge-L), 4.91% (MToken), and 2.73% (MLine) on average.

Time Efficiency. For the training hours, the time cost of SCPatcher is 11 hours, which is only longer than the original VulSlicer and DeepBalance. Combining both the comparison results and time efficiency, we believe that SCPatcher has advantages over baselines on extracting coding examples.

Benefits. We believe that the benefits of SCPatcher come from the following four aspects: 1) The area extraction first limit the approximate range of coding examples, thus reducing the impact of noisy codes. Not only the SCPatcher, but also VulSlicer, DeepBalance, and GPT-3 benefit from the areas according to the experiment results. 2) The SFV contains the high-level representation of security-related features, thus facilitating the extraction of coding examples. The performances of VulSlicer and DeepBalance are also improved after combining the area and SFV. 3) The Attention-based Selector can effectively learn the external knowledge with little parameters, thus reducing the training hours. 4) The hierarchical slicer can locate the coding examples from both `function` and `fragment`, thus improving the accuracy.

Answering RQ1*: SCPatcher outperforms all the baseline on extracting coding examples. It reaches the highest Rouge-L, MToken, and MLine with 63.33%, 74.23%, and 72.38%, outperforming the best baseline with 4.03% (Rouge-L), 4.91% (MToken), and 2.73% (MLine).*

B. Performance on Extracting the Coding Explanations

Table V illustrates the comparison results of SCPatcher on extracting the coding explanations between baselines, and the best performance is highlighted with **bold face**.

Comparison with Original Baselines. Comparing the results with the original baselines, we can see that, SCPatcher outperforms these baselines on average, improving the best original baseline GPT-3 by 11.78% (Precision), 11.83% (Recall), and 11.81% (F1) on average.

Comparison with Enhanced Baselines. Comparing the results with enhanced baselines, we can see that, SCPatcher outperforms these baselines on average, improving the best enhanced baseline GPT3+Area by 4.04% (Precision), 3.91% (Recall), and 3.97% (F1) on average.

Time Efficiency. For the training hours, the time cost of SCPatcher is 11 hours, which is only longer than the original BERTSum. Combining both the comparison results and time efficiency, we believe that SCPatcher has advantages over baselines on extracting coding explanations.

Benefits. We believe that the benefits also come from the areas, SFV, and the Attention-based Selector. The performances of baselines benefit from the area and SFV, with over 10% improvements in Precision, Recall, and F1.

Answering RQ2*: SCPatcher outperforms all the baseline on extracting the coding explanations. It reaches the highest Precision, Recall and F1 with 79.43%, 79.70%, and 79.56%, outperforming the best baseline GPT-3+Area with 4.04% (Precision), 3.91% (Recall), and 3.97% (F1).*

C. Performance of Matching the CWE

Table VI shows the matching results of CWEs from the 447 SCP specifications. We can see that, there are 409 SCP specifications are matched with existing CWEs, while 38 posts are unmatched. Some of these unmatched SCP specifications may be incorrect predictions due to the limitation of SCPatcher, while the other part may contain weaknesses that are not currently included in CWE.

Fig. 4 shows the number of LOC and sentences of SPC specifications for each CWE. We can see that, for each CWE, our approach can extract insecure/secure code examples and their explanations. Among them, the insecure code examples account for the most with an average of 107 LOC, while the secure coding explanation account for the least with an average of 66 sentences.

TABLE VI: The size of matched and unmatched CWE.

Items		#SCP	#LOC	#Sentences
CWE	Matched	409	3,431	2,074
	Unmatched	38	218	239
Public SCP	Matched	392	3,074	1,967
	Unmatched	55	575	346

TABLE VII: The performances on matching the CWE.

CWE	#Pred	Pre. (%)	CWE	#Pred	Pre. (%)
CWE-787	76	81.58	CWE-138	5	80.00
CWE-79	68	70.59	CWE-183	5	80.00
CWE-78	58	79.31	CWE-261	4	100.00
CWE-352	46	69.64	CWE-5	3	66.67
CWE-190	40	87.50	CWE-23	3	100.00
CWE-287	37	86.49	CWE-125	3	66.67
CWE-94	27	77.78	CWE-603	3	66.67
CWE-91	10	80.00	CWE-1047	3	100.00
CWE-66	9	77.78	CWE-1065	2	100.00
CWE-121	7	71.43	***Average***		***81.43***

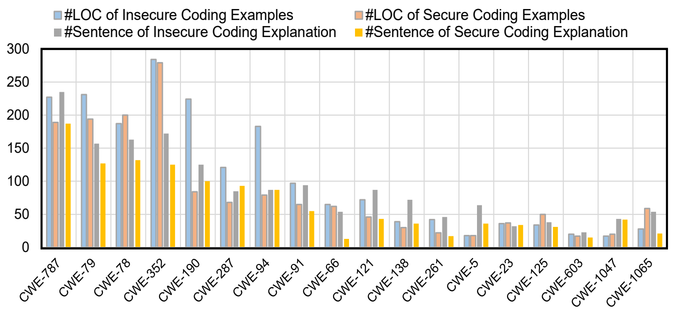


Fig. 4: The statistics of SPC specification for individual CWE.

Table VII shows the performances on matching the CWE. The **#Pred** column shows the number of predictions. The **Pre (%)** column shows the precision performance. We can see that, the number of predictions shows a long-tail trend. The top-3 maximum number of predictions are CWE-787 (with 76 SCP specifications), CWE-79 (68 SCP specifications), and CWE-78 (58 SCP specifications). The average Precision of matching the CWEs is 81.43%. For each CWE-ID, 15/19 has over 70% Precision. These results indicate that the SCPatcher can satisfactorily match the CWE for most of the time.

Answering RQ3*: SCPatcher can accurately matches the CWE with 81.43% (Precision) on average.*

D. Performance of Matching the Public SCP

Table VI shows the matching results of public SCP from the 447 SCP specifications. We can see that, there are 392 SCP specifications are matched with existing OWASP types, while 55 posts are unmatched. These unmatched SCP specifications may be incorrect predictions, or the novel SCPs that have not been included by OWASP.

Fig. 5 shows the number of LOC and sentences of SPC specifications for each OWASP type. Overall, the SCPatcher has the ability to enrich the public SCP with various information. We can see that, for each OWASP type, our approach can also extract insecure/secure code examples and their explanations. Among them, the insecure coding examples account for the

TABLE VIII: The performances on matching the public SCP.

OWASP Types	#Pred	Pre. (%)	OWASP Types	#Pred	Pre. (%)
Input Validation	44	84.09	Password Manage.	12	83.33
Session Manage.	57	87.72	Output Encoding	7	71.42
Access Control	53	75.47	Crypto. Practices	7	85.71
Database Security	37	89.19	Error Handling	6	83.33
Data Protection	33	87.88	Community Security	5	60.00
System Config.	31	80.65	General SCPs	4	50.00
File Manage.	50	76.00	***Average***		***78.34***
Memory Manage.	46	80.43			

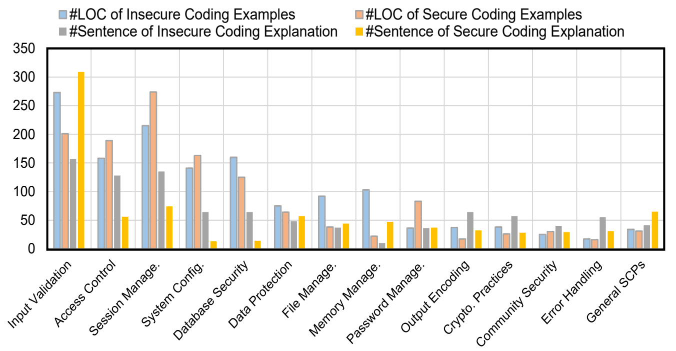


Fig. 5: The statistics of SCP specification for individual OWASP's public SCP.

most with an average 100 LOC, while the secure coding explanation account for the least with an average 59 sentences.

Table VIII shows the performances on matching the public SCP. We can see that, the number of predictions shows a long-tail trend. The top-3 maximum number of predictions is Session Management (with 57 SCP specifications), Access Control (53 SCP specifications), and File Management (46 SCP specifications). The average Precision of matching the public SCP is 78.34%. For each OWASP type, 12/14 has over 70% Precision. These results indicate that the SCPatcher can satisfactorily match the public SCP for most of the time.

Answering RQ4: *SCPatcher can enrich the SCP with 392 SCP specifications, 3,074 LOC, and 1,967 sentences. It accurately matches the public SCP with 78.34% (Precision) on average.*

VI. Discussion

A. Effect of Prompt Template

To analyze the performances of different prompt templates, we compare the performances of four templates, as is shown in Table IX. Two of them are Prefix Templates [49] with the Q&A format. Another two templates are Cloze Templates [13] that utilize the cloze-testing to generate the output sentences. Since the minor differences in the prompt template may greatly affect the performances of SCPatcher [15], we only replace the "code and sentence" in the original template with "coding examples and explanations", and compare their performances on the SCP specification extraction.

Fig. 6 shows the results of different prompt templates. We can see that, for the comparison between template types,

TABLE IX: The performances on different prompt templates.

Template Types	ID	Template Text
Prefix Template	Prefix 1	[X] What are {insecure|secure} coding examples and explanations? [Y]
	Prefix 2	[X] What are {insecure|secure} codes and sentences? [Y]
Cloze Template	Cloze 1	[X] The {insecure|secure} coding examples and explanations [Y].
	Cloze 2	[X] The {insecure|secure} codes and sentences are [Y].

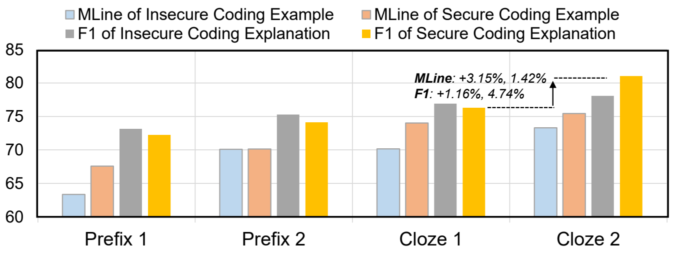


Fig. 6: The results of prompt template comparison (%).

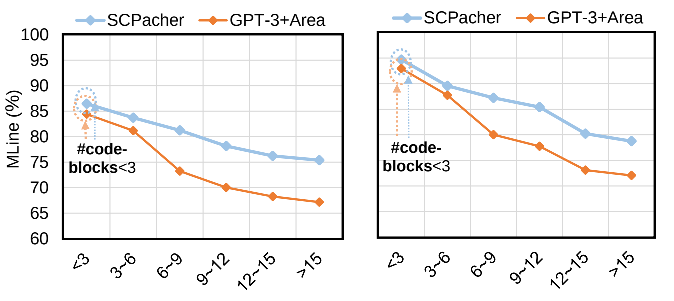
the cloze templates outperforms the prefix templates with 5.02% MLine on Insecure Coding Example, 5.88% MLine on Secure Coding Example, 3.28% F1 on Insecure Coding Explanation, and 5.48% F1 on Secure Coding Explanation. For the comparison on all the four templates, our template (Cloze 2) achieves the highest MLine and F1, outperforming the rest templates with 3.15% MLine on Insecure Coding Example, 1.42% MLine on Secure Coding Example, 1.16% F1 on Insecure Coding Explanation, and 4.74% F1 on Secure Coding Explanation. In summary, our prompt template can effectively help SCPatcher on extracting the coding examples and explanations from security posts.

B. Effect of Sentence and Code-Block Numbers

To analyze the performances of SCPatcher on numbers of sentences and code-blocks, we compare the performances between SCPatcher and the SOTA baseline, i.e., the GPT-3+Area, on several number intervals. To determine the number intervals, we manually inspect the testing dataset, then set the intervals of code blocks as "<3" to ">15", and set the number intervals of sentences from "5∼10" to ">30". Each interval contains a similar number of posts.

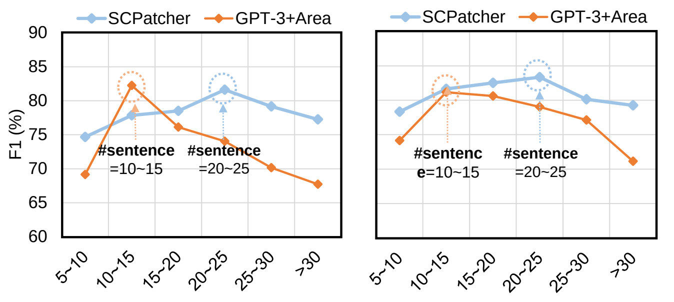
Fig. 7 and 8 illustrate the effect of code-block and sentence numbers. We can see that, for the comparison of LOC intervals in Fig. 7 (a) and (b), the SCPatcher outperforms the GPT3-Area with the average 6.15% MLine on Insecure Coding Example and 5.37% on Secure Coding Example. For the comparison of sentence intervals in Fig. 8 (a) and (b), the SCPatcher outperforms the GPT-3+Area with the average 4.94% F1 on Insecure Coding Explanation, and 3.67% F1 on Secure Coding Explanation.

For the difference between the highest and lowest MLine on extracting coding examples, and F1 on extracting coding explanations, we can see that, the fluctuation of SCPatcher is smaller than GTP3+Area, with 6.20% MLine on Insecure Coding Example, 4.95% MLine on Secure Coding Example,



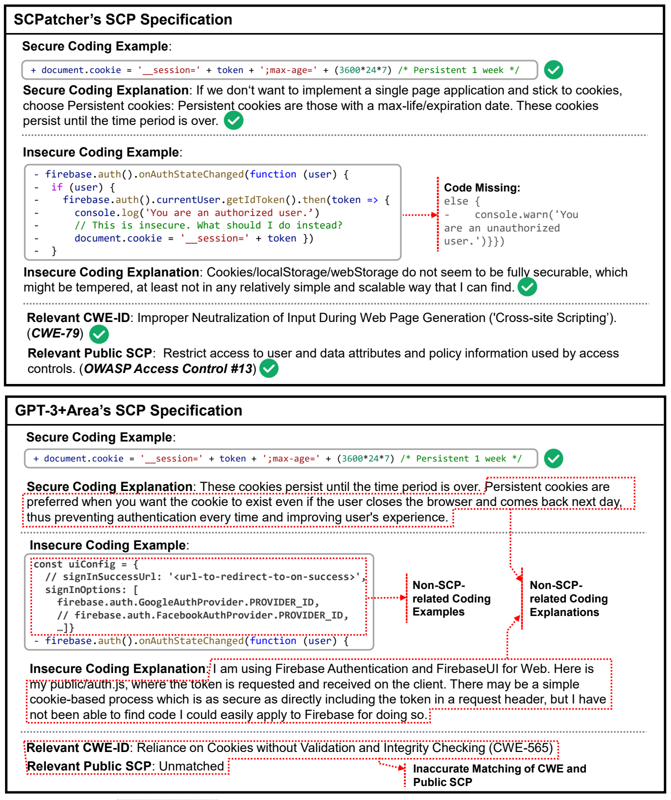
(a) Effect of #code-block on extracting insecure coding examples. (b) Effect of #code-block on extracting secure coding examples.

Fig. 7: Effect of #code-block on extracting coding examples.



(a) Effect of #sentence on extracting insecure coding explanations. (b) Effect of #sentence on extracting secure coding explanations.

Fig. 8: Effect of #sentence on extracting coding explanations.



SCPatcher's SCP Specification

Secure Coding Example:

```
+ document.cookie = '__session=' + token + ';max-age=' + (3600*24*7) /* Persistent 1 week */
```
✓

Secure Coding Explanation: If we don't want to implement a single page application and stick to cookies, choose Persistent cookies: Persistent cookies are those with a max-life/expiration date. These cookies persist until the time period is over. ✓

Insecure Coding Example:

```
- firebase.auth().onAuthStateChanged(function (user) {
-  if (user) {
-    firebase.auth().currentUser.getIdToken().then(token => {
-      console.log('You are an authorized user.')
-      // This is insecure. What should I do instead?
-      document.cookie = '__session=' + token })
-  }
```

Code Missing:

```
else {
-    console.warn('You
are an unauthorized
user.')}})
```

Insecure Coding Explanation: Cookies/localStorage/webStorage do not seem to be fully securable, which might be tempered, at least not in any relatively simple and scalable way that I can find. ✓

Relevant CWE-ID: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'). (***CWE-79***) ✓

Relevant Public SCP: Restrict access to user and data attributes and policy information used by access controls. (***OWASP Access Control #13***) ✓

GPT-3+Area's SCP Specification

Secure Coding Example:

```
+ document.cookie = '__session=' + token + ';max-age=' + (3600*24*7) /* Persistent 1 week */
```
✓

Secure Coding Explanation: These cookies persist until the time period is over. Persistent cookies are preferred when you want the cookie to exist even if the user closes the browser and comes back next day, thus preventing authentication every time and improving user's experience.

Insecure Coding Example:

```
const uiConfig = {
  // signInSuccessUrl: '<url-to-redirect-to-on-success>',
  signInOptions: [
    firebase.auth.GoogleAuthProvider.PROVIDER_ID,
    // firebase.auth.FacebookAuthProvider.PROVIDER_ID,
    …]}
- firebase.auth().onAuthStateChanged(function (user) {
```

Non-SCP-related Coding Examples

Non-SCP-related Coding Explanations

Insecure Coding Explanation: I am using Firebase Authentication and FirebaseUI for Web. Here is my public/auth.js, where the token is requested and received on the client. There may be a simple cookie-based process which is as secure as directly including the token in a request header, but I have not been able to find code I could easily apply to Firebase for doing so.

Relevant CWE-ID: Reliance on Cookies without Validation and Integrity Checking (CWE-565)

Relevant Public SCP: Unmatched

Inaccurate Matching of CWE and Public SCP

*** The text with red dashed box indicates the incorrect extraction result.**

Fig. 9: The results of SCP specification extraction on the motivating example (Fig 1).

7.97% F1 on Insecure Coding Explanation, and 5.02% F1 on Secure Coding Explanation.

In summary, SCPatcher performs well on security posts with different sentence and code block numbers.

C. Qualitative Evaluation

To qualitatively evaluate the performance of SCPatcher on extracting the SCP specifications, we first conduct a case study to compare the extraction results between SCPatcher and GPT-3+Area, and then we analyze the cases where the CWE and public SCP are not matched successfully.

We compare SCPatcher with the SOTA baseline, i.e., GPT-3+Area, on extracting SCP specifications on the motivating example in Fig. 1. Since GPT-3+Area does not support matching of the CWE and public SCP, we first extract the coding examples and explanations with GPT-3+Area and then utilize the same security similarity to match the CWE and public SCP in Section III-C. Fig. 9 shows the results. We find that SCPatcher can accurately extract the coding examples and explanations and successfully matches the CWE and public SCP. The GPT-3+Area, on the contrary, introduces incorrect code explanations and irrelevant lines of code and thus matches the incorrect CWE and public SCP. This case study shows that SCPatcher is more accurate than the SOTA baseline in extracting SCP specifications.

Although SCPatcher can accurately extract the SCP specifications in most cases, SCPatcher cannot successfully match the CWE and public SCP for around 10% of the extracted SCP specifications in our experiment. We manually inspect these SCP specifications and find the following three reasons:

- **Non-SCP-related Code (6.2%).** The code examples in the security posts are not related to the SCPs, so SCPatcher cannot match them to the CWEs and public SCPs.
- **Inaccurate Extraction (2.4%).** Due to the limitation of SCPatcher, some extracted SCP specifications are inaccurate and thus cannot be correctly matched to the CWEs and public SCPs.
- **Potential New Public SCPs (1.4%).** Some SCP specifications may incorporate new public SCPs that have not been incorporated by the OWASP. For example, the post #72865733 proposes a novel SCP in Jul, 2022, indicating that developers can utilize the Web Application Firewall (WAF) to enhance the security of the K8s cloud systems. However, this SCP has not been incorporated by OWASP.

D. Threat to Validity

There are internal, external, and constructive threats that may affect the performances of SCPatcher.

Internal Threats. The first internal threat lies in the data augmentation strategy we use. We augment the dataset from 3,907 posts to 11,091 posts using EDA augmentation. The resulting augmented dataset may have semantic differences from the original dataset. To alleviate it, we manually inspect 100 augmented samples and find that 94% of inspected samples are correct, and only 6% of samples have slight differences. The second internal threat comes from dataset collection and preprocessing. We only collect the posts with code blocks, even though some posts without code blocks may have also been discussed on SCP specifications. To alleviate it, we manually inspect 100 security posts that are filtered in our data preparation, and find that less than 3% posts without <code> tags have SCP specifications, so the effect of this internal threat is small.

External Threat. The external threat may come from multi-topic security posts. Due to the open and crowded nature of Stack Overflow security posts, it is inevitable that developers may discuss multiple topics in one post. For example, developers may post similar insecure practices for the questioners to refer to, as well as their solutions for questioners' security issues. Since we only extract secure coding examples and explanations from the Answer part, we would miss some insecure coding examples and explanations from that part. In future work, we plan to improve the area extraction to better identify the areas in the post where the secure/insecure coding examples and explanations may occur.

Constructive Threat. The constructive threat mainly comes from the metrics we use. We choose Rouge-L, MToken, and MLine to evaluate the extraction of coding examples. We also choose Precision, Recall, and F1 to evaluate the performances on extracting the coding explanations. We manually label the LOC of coding examples, and the sentences of coding explanations as the ground-truth while calculating these metrics. This threat is mitigated by the fact that all the posts are reviewed and discussed by the annotators when labeling the coding examples and explanations.

VII. Related Works

Our work is related to the prior studies on analyzing secure coding practices and analyzing crowd security discussions.

Analysis of Secure Coding Practices. The discussions on secure coding practices have kept increasing in recent years. Many works have been proposed to analyze secure coding practices. Some works are focused on how to teach SCPs. For example, Chi et al. [50] propose a method for teaching SCPs on STEM students. Singleton et al. [51] propose CryptoTutor, a novel method for teaching SCP with Misuse Pattern Detection. Some other works are focused on applying SCPs in software development. For example, Khalili et al. [52] perform a case study on analyzing the SCPs in Industrial Control Systems (ICS) by manually monitoring the real-world industrial application. Meng et al. [53] analyze the SCPs in Java programming, especially the Spring security and its challenges. Anis et al. [54] propose a system to verify the development of web applications with SCPs, and apply the system on JavaScript applications. While SCPs are useful, they can be difficult to follow in practice due to their lack of details; thus, some previous works have proposed to enrich SCPs. Khalili et al. [52] manually provide the details of ICS SCPs from real-world industrial application. Gorski et al. [55] mines the security-related information in non-security API documents to support SCPs. Different from these SCP enrichment works, this work is the first to automatically mine crowd security discussions to extract the SCP specifications.

Analysis of Crowd Security Discussions. The crowd security discussions in online knowledge-sharing platforms have recently been studied by researchers. Some researchers are interested in the linguistic features of security discussions. Meyers et al. [56] studied security discussions from their formality, informativeness, implicature, politeness, and uncertainty. Some other researchers are interested in summarizing the main topics of security discussions. Yang et al. [57] study security concerns on the following topics: web security, mobile security, cryptography, software security, and system security, and analyze each topic's popularity. Zahedi et al. [58] utilize a qualitative analysis to summarize 26 topics on GitHub, and extract the frequently occurring topics, such as "login", "hash", and "password" etc.. Le et al. [59] applly topic a modeling method, i.e., LDA, to identify 13 main security-related topics on Stack Overflow. These studies are mainly concerned with the characteristics of security discussions via topic modeling and qualitative analysis. Different from them, our work focuses on extracting the coding examples and explanations from the crowd security discussions, so as to enrich public SCPs.

VIII. Conclusion and Future Work

In this paper, we introduce the SCPatcher, which is an automated approach to enrich secure coding practices by mining crowd security discussions. SCPatcher first extracts the area of coding examples and coding explanations with a fix-prompt tuned LLM via Prompt Learning, Then, it hierarchically slices the lengthy code to the coding examples and summarizes the coding explanations based on the areas, Finally, SCPatcher matches the CWE and Public SCP, and integrates them with extracted coding examples and explanations to form the SCP specifications. We collect the 3,907 security posts from Stack Overflow, and augment it to the 11,091 posts. The experimental results show that SCPatcher outperforms all baselines in extracting the coding examples with 2.73% MLine on average, as well as coding explanations with 3.97% F1 on average. Moreover, we apply SCPatcher on 447 posts to further evaluate its practicality. The extracted SCP specifications enrich the public SCP with 3,074 LOC and 1,967 sentences.

In the future, we plan to further improve our approach with more extended datasets from other knowledge-sharing platforms. We also plan to enrich more public SCPs, such as those proposed by Google and UC Berkeley, to further evaluate the practicality of SCPatcher.

Acknowledgments

We sincerely appreciate anonymous reviewers for their constructive and insightful suggestions for improving this manuscript. This work is supported by the National Natural Science Foundation of China Grant No. 62272445, 62232016, 62072442, the Youth Innovation Promotion Association Chinese Academy of Sciences, and the University of Queensland NSRSG Grant NS-2201.

References

[1] Y. Acar, C. Stransky, D. Wermke, C. Weir, M. L. Mazurek, and S. Fahl, "Developers need support, too: A survey of security advice for software developers," in *IEEE Cybersecurity Development, SecDev 2017, Cambridge, MA, USA, September 24-26, 2017*. IEEE Computer Society, 2017, pp. 22–26.

[2] Google, "Best Practices for Privacy and Security," https://developers.google.cn/assistant/sdk/guides/library/python/best-practices/privacy-and-security, 2023.

[3] U. Berkeley, "Secure Coding Practice Guidelines," https://security.berkeley.edu/secure-coding-practice-guidelines, 2023.

[4] OWASP, "Secure Coding Practices Quick Reference Guide," https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v1-1b.pdf, 2022.

[5] S. Overflow, "Stack Overflow - Where Developers Learn, Share, & Build Careers," https://stackoverflow.com/, 2022.

[6] F. van Puffelen, "Best Practices for Firebase Data Models," https://twitter.com/puf/status/1483457400606580736, 2022.

[7] *Replication Package for "SCPatcher: Mining Crowd Security Discussions to Enrich Secure Coding Practices"*. Zenodo, Aug. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8254682

[8] L. Shi, Z. Jiang, Y. Yang, X. Chen, Y. Zhang, F. Mu, H. Jiang, and Q. Wang, "ISPY: automatic issue-solution pair extraction from community live chats," in *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 2021, pp. 142–154.

[9] M.Corporation, "Common weakness enumeration," https://cwe.mitre.org/, 2021.

[10] T. Schick and H. Schütze, "Few-shot text generation with natural language instructions," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 390–402.

[11] 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.

[12] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, "Parameter-efficient transfer learning for nlp," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2790–2799.

[13] F. Petroni, T. Rocktäschel, S. Riedel, P. Lewis, A. Bakhtin, Y. Wu, and A. 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)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 2463–2473.

[14] 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 (Volume 1: Long Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 4582–4597.

[15] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang, "GPT understands, too," *CoRR*, vol. abs/2103.10385, 2021.

[16] L. Adolphs, T. Gao, J. Xu, K. Shuster, S. Sukhbaatar, and J. Weston, "The CRINGE loss: Learning what language not to model," *CoRR*, vol. abs/2211.05826, 2022.

[17] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding," in *NAACL-HLT*. Association for Computational Linguistics, 2019, pp. 4171–4186.

[18] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[19] G. Betz, K. Richardson, and C. Voigt, "Thinking aloud: Dynamic context generation improves zero-shot reasoning performance of GPT-2," *CoRR*, vol. abs/2103.13033, 2021.

[20] 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," *J. Mach. Learn. Res.*, vol. 21, pp. 140:1–140:67, 2020.

[21] K. M. Yoo, D. Park, J. Kang, S. Lee, and W. Park, "Gpt3mix: Leveraging large-scale language models for text augmentation," in *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*. Association for Computational Linguistics, 2021, pp. 2225–2239.

[22] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[23] S. Pan, J. Zhou, F. R. Côgo, X. Xia, L. Bao, X. Hu, S. Li, and A. E. Hassan, "Automated unearthing of dangerous issue reports," 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. 834–846.

[24] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. ACL, 2014, pp. 1532–1543.

[25] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, "Recurrent models of visual attention," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 2014, pp. 2204–2212.

[26] K. Margatina, C. Baziotis, and A. Potamianos, "Attention-based conditioning methods for external knowledge integration," in *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics, 2019, pp. 3944–3951.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 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. 5998–6008.

[28] T. Lopez, T. T. Tun, A. K. Bandara, M. Levine, B. Nuseibeh, and H. Sharp, "An anatomy of security conversations in stack overflow," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. ACM, 2019, pp. 31–40.

[29] ——, "An investigation of security conversations in stack overflow: perceptions of security and community involvement," in *Proceedings of the 1st International Workshop on Security Awareness from Design to Deployment, SEAD@ICSE 2018, Gothenburg, Sweden, May 27, 2018*. ACM, 2018, pp. 26–32.

[30] Y. Hu, H. Jiang, and Z. Hu, "Measuring code maintainability with deep neural networks," *Frontiers Comput. Sci.*, vol. 17, no. 6, p. 176214, 2023.

[31] E. Shi, Y. Wang, L. Du, H. Zhang, S. Han, D. Zhang, and H. Sun, "CAST: enhancing code summarization with hierarchical splitting and reconstruction of abstract syntax trees," 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. 4053–4062.

[32] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. IEEE / ACM, 2019, pp. 783–794.

[33] Python, "pyparser 0.1," https://pypi.org/project/pyparser/0.1/, 2014.

[34] C. Sun and Z. Yang, "Transfer learning in biomedical named entity recognition: An evaluation of BERT in the pharmaconer task," in *Proceedings of The 5th Workshop on BioNLP Open Shared Tasks, BioNLP-OST@EMNLP-IJNCLP 2019, Hong Kong, China, November 4, 2019*. Association for Computational Linguistics, 2019, pp. 100–104.

[35] H. Tayyar Madabushi, E. Kochkina, and M. Castelle, "Cost-Sensitive BERT for Generalisable Sentence Classification on Imbalanced Data,"

in *Proceedings of the Second Workshop on Natural Language Processing for Internet Freedom: Censorship, Disinformation, and Propaganda*. Association for Computational Linguistics, Nov. 2019, pp. 125–134.

[36] T. H. M. Le, R. Croft, D. Hin, and M. A. Babar, "Demystifying the Mysteries of Security Vulnerability Discussions on Developer Q&A Sites," *CoRR*, vol. abs/2008.04176, 2020.

[37] S. Exchange, "Stack Exchange Sites," https://stackexchange.com/sites, 2022.

[38] X. Yang, D. Lo, X. Xia, Z. Wan, and J. Sun, "What security questions do developers ask? A large-scale study of stack overflow posts," *J. Comput. Sci. Technol.*, vol. 31, no. 5, pp. 910–924, 2016.

[39] J. E. Pérez, J. Díaz, J. G. Martin, and B. Tabuenca, "Systematic Literature Reviews in Software Engineering - Enhancement of the Study Selection Process Using Cohen's Kappa Statistic," *J. Syst. Softw.*, vol. 168, p. 110657, 2020.

[40] J. W. Wei and K. Zou, "EDA: easy data augmentation techniques for boosting performance on text classification tasks," 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. 6381–6387.

[41] S. Salimi and M. Kharrazi, "Vulslicer: Vulnerability detection through code slicing," *J. Syst. Softw.*, vol. 193, p. 111450, 2022.

[42] S. Liu, G. Lin, Q. Han, S. Wen, J. Zhang, and Y. Xiang, "Deepbalance: Deep-learning and fuzzy oversampling for vulnerability detection," *IEEE Trans. Fuzzy Syst.*, vol. 28, no. 7, pp. 1329–1343, 2020.

[43] M. T. B. Nazim, M. J. H. Faruk, H. Shahriar, M. A. Khan, M. Masum, N. Sakib, and F. Wu, "Systematic analysis of deep learning model for vulnerable code detection," in *46th IEEE Annual Computers, Software, and Applications Conferenc, COMPSAC 2022, Los Alamitos, CA, USA, June 27 - July 1, 2022*. IEEE, 2022, pp. 1768–1773.

[44] Y. Liu and M. Lapata, "Text summarization with pretrained encoders," 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. 3728–3738.

[45] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Association for Computational Linguistics, 2020, pp. 7871–7880.

[46] C.-Y. Lin, "Rouge: a package for automatic evaluation of summaries," in *Workshop on Text Summarization Branches Out, Post-Conference Workshop of ACL 2004, Barcelona, Spain*, July 2004, pp. 74–81.

[47] V. Cochard, D. Pfammatter, C. T. Duong, and M. Humbert, "Investigating graph embedding methods for cross-platform binary code similarity detection," in *7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022*. IEEE, 2022, pp. 60–73.

[48] S. M. LaValle, M. S. Branicky, and S. R. Lindemann, "On the relationship between classical grid search and probabilistic roadmaps," *Int. J. Robotics Res.*, vol. 23, no. 7-8, pp. 673–692, 2004.

[49] 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.

[50] H. Chi, E. L. Jones, and J. Brown, "Teaching secure coding practices to STEM students," in *Proceedings of the 2013 Information Security Curriculum Development Conference, InfoSecCD 2013, Kennesaw, Georgia, USA, October 12, 2013*. ACM, 2013, pp. 42–48.

[51] L. Singleton, R. Zhao, M. Song, and H. P. Siy, "Cryptotutor: Teaching secure coding practices through misuse pattern detection," in *SIGITE '20: The 21st Annual Conference on Information Technology Education, Virtual Event, USA, October 7-9, 2020*. ACM, 2020, pp. 403–408.

[52] A. Khalili, A. Sami, M. Azimi, S. Moshtari, Z. Salehi, M. Ghiasi, and A. A. Safavi, "Employing secure coding practices into industrial applications: a case study," *Empir. Softw. Eng.*, vol. 21, no. 1, pp. 4–16, 2016.

[53] N. Meng, S. Nagy, D. D. Yao, W. Zhuang, and G. A. Arango-Argoty, "Secure coding practices in java: challenges and vulnerabilities," in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. ACM, 2018, pp. 372–383.

[54] A. Anis, M. Zulkernine, S. Iqbal, C. Liem, and C. Chambers, "Securing web applications with secure coding practices and integrity verification," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress, DASC/PiCom/DataCom/CyberSciTech 2018, Athens, Greece, August 12-15, 2018*. IEEE Computer Society, 2018, pp. 618–625.

[55] P. L. Gorski, S. Möller, S. Wiefling, and L. L. Iacono, ""i just looked for the solution!"on integrating security-relevant information in non-security API documentation to support secure coding practices," *IEEE Trans. Software Eng.*, vol. 48, no. 9, pp. 3467–3484, 2022.

[56] B. S. Meyers, N. Munaiah, A. Meneely, and E. Prud'hommeaux, "Pragmatic characteristics of security conversations: an exploratory linguistic analysis," in *Proceedings of the 12th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE@ICSE 2019, Montréal, QC, Canada, 27 May 2019*. IEEE / ACM, 2019, pp. 79–82.

[57] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun, "What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts," *Journal of Computer Science and Technology*, vol. 31, no. 5, pp. 910–924, 2016.

[58] M. Zahedi, M. A. Babar, and C. Treude, "An empirical study of security issues posted in open source projects," in *51st Hawaii International Conference on System Sciences, HICSS 2018, Hilton Waikoloa Village, Hawaii, USA, January 3-6, 2018*. ScholarSpace / AIS Electronic Library (AISeL), 2018, pp. 1–10.

[59] T. H. M. Le, R. Croft, D. Hin, and M. A. Babar, "A large-scale study of security vulnerability support on developer q&a websites," in *EASE 2021: Evaluation and Assessment in Software Engineering, Trondheim, Norway, June 21-24, 2021*. ACM, 2021, pp. 109–118.