APICom: Automatic API Completion via Prompt Learning and Adversarial Training-based Data Augmentation

Yafeng Gu
Nantong University
China
yafeng.g@outlook.com

Yiheng Shen
Nantong University
China
yiheng.s@outlook.com

Xiang Chen*
Nantong University
China
xchencs@ntu.edu.cn

Shaoyu Yang
Nantong University
China
shaoyuyoung@gmail.com

Yiling Huang
Nantong University
China
hylwing13150522911@163.com

Zhixiang Cao
Nantong University
China
486478817@qq.com

ABSTRACT

Based on developer needs and usage scenarios, API (Application Programming Interface) recommendation is the process of assisting developers in finding the required API among numerous candidate APIs. Previous studies mainly modeled API recommendation as the recommendation task, which can recommend multiple candidate APIs for the given query, and developers may not yet be able to find what they need. Motivated by the neural machine translation research domain, we can model this problem as the generation task, which aims to directly generate the required API for the developer query. After our preliminary investigation, we find the performance of this intuitive approach is not promising. The reason is that there exists an error when generating the prefixes of the API. However, developers may know certain API prefix information during actual development in most cases. Therefore, we model this problem as the automatic completion task and propose a novel approach APICom based on prompt learning, which can generate API related to the query according to the prompts (i.e., API prefix information). Moreover, the effectiveness of APICom highly depends on the quality of the training dataset. In this study, we further design a novel gradient-based adversarial training method ATCom for data augmentation, which can improve the normalized stability when generating adversarial examples. To evaluate the effectiveness of APICom, we consider a corpus of 33k developer queries and corresponding APIs. Compared with the state-of-the-art baselines, our experimental results show that APICom can outperform all baselines by at least 40.02%, 13.20%, and 16.31% in terms of the performance measures EM@1, MRR, and MAP. Finally, our ablation studies confirm the effectiveness of our component setting (such as our designed adversarial training method, our used pre-trained model, and prompt learning) in APICom.

*Xiang Chen is the corresponding author.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**.

KEYWORDS

API Completion, API Recommendation, Adversarial Training, Prompt Learning, Pre-trained Model

ACM Reference Format:
Yafeng Gu, Yiheng Shen, Xiang Chen, Shaoyu Yang, Yiling Huang, and Zhixiang Cao. 2023. APICom: Automatic API Completion via Prompt Learning and Adversarial Training-based Data Augmentation. In *Proceedings of 14th Asia-Pacific Symposium on Internetware (Internetware 2023).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

API can provide developers with the ability to utilize pre-built functions and integrate disparate software systems, which can improve the efficiency and effectiveness of software development and maintenance. Therefore, API (Application Programming Interface) has become an important component of modern software development. However, with the rapid growth of available APIs and the increasing complexity of software systems, it has become difficult for developers to find the most suitable APIs based on their needs. Therefore, effective API recommendation approaches can help to improve the efficiency and effectiveness of software development and maintenance.

According to our statistical analysis, 10,870 posts can be searched by using the keyword "API" on Stack Overflow until March 2023. In a post[1] shown in Table 1, the developer wants to find a corresponding API in the Grails plug-in or Java library, which can generate PDF417 images in the Grails application and send them by email. This post receives more than 12.8k likes, and the first answer about API receives more than 75.2k likes. Based on the above analysis, we can find it challenging for developers who are unfamiliar with the development task at hand to find the required API quickly.

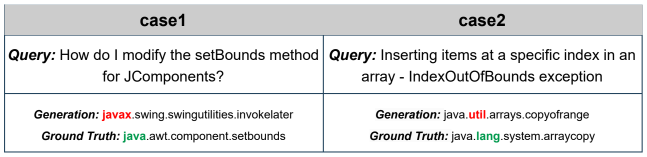
To solve this problem, automatic API recommendation approaches have been studied. Most of the previous studies mainly modeled API recommendation as the information retrieval (IR) task, which can recommend multiple candidate APIs based on text similarity. However, the developers still need to select the

[1]https://stackoverflow.com/questions/5671386/pdf417-image-generation-api-recommendation

Table 1: A post related to API recommendation from Stack Overflow

Post Title	PDF417 image generation API recommendation
Post Content	in a grails application I need to generate a PDF417 image and send it via email. Can anybody recommend me a (hopefully free) Grails plugin or Java library?
Post Tags	java , grails , barcode

desired API from these candidate APIs, and sometimes even the correct API is not included in these candidate APIs. Motivated by the neural machine translation research domain, one intuitive way is to model this problem as the automatic generation task, which can generate the required API for the developer's query directly. Although the popular generation models perform well in software engineering generation tasks (such as source code summarization [1, 20, 21, 45, 49], issue title generation [5, 22], code generation [44, 47], Stack Overflow title generation [23, 51]), we find that the performance of this intuitive way is not promising after our preliminary investigation. Figure 1 shows two examples of generating incorrect APIs by using this intuitive generation approach. In the first example, the ground truth is "java.awt.component.setbounds". If we use the intuitive generation approach, the generated API is "java.swing.swingutilities.invokelater". In this case, there is an error with the first prefix. In the second example, the ground truth is "java.lang.system.arraycopy". If we use the intuitive generation approach, the generated API is "java.util.arrays.copyofrange". In this case, there is an error with the second prefix. Based on these two examples, we find that incorrectly generated prefixes may cause the low performance of this intuitive generation approach. However, based on our observation of practical software development and maintenance, developers may remember certain API prefix information and it is the remaining parts of the API that they really need to turn to for help.



case1	case2
Query: How do I modify the setBounds method for JComponents?	***Query:*** Inserting items at a specific index in an array - IndexOutOfBounds exception
Generation: javax.swing.swingutilities.invokelater ***Ground Truth:*** java.awt.component.setbounds	***Generation:*** java.util.arrays.copyofrange ***Ground Truth:*** java.lang.system.arraycopy

Figure 1: Two examples to show the limitations of using the intuitive generation approach for API recommendation

Based on the above motivation, we are the first to model the API recommendation problem into the API completion task and then propose a novel approach APICom. In particular, developers only need to input the description of the encountered problem (i.e., query) and certain API prefixes they know. After that, the corresponding correct API may be generated. To better complete the API, we consider prompt learning, which can predict the content of MASK (i.e., the remaining parts of the API) more accurately according to limited prompts. Specifically, prompt learning is a technique that provides a small amount of context to a language model to guide its predictions. In our API completion task, prompt learning can improve performance and prevent catastrophic forgetting through generated task-specific prompts.

On the other hand, the performance of APICom highly depends on the quality of the corpus used for training. However, collecting such data manually can be time-consuming and labor-intensive. To alleviate this problem, we propose a novel adversarial training method ATCom, which can improve the stability of the normalization process while retaining the advantages of L_1 and L_2 normalization.

To evaluate the effectiveness of APICom, We consider a high-quality corpus shared by Huang et al. [13], which contains API queries and corresponding APIs. To generate the prompts, we randomly mask a certain number of words (separated by dots) at the end of the APIs. Then, ATCom generates K adversarial examples for all the inputs to augment the training set. Finally, we fine-tune a pre-trained model CodeT5 [38] to learn the potential API completion patterns. Since most of the previous studies [13, 16, 32, 40, 50] mainly modeled API recommendation as the recommendation task, we first consider MRR (Mean Reciprocal Rank) [31] and MAP (Mean Average Precision) [34] as model performance evaluation measures. Moreover, since we model this problem as an automatic completion task in this study. we second consider EM (Exact Match) [7, 15] measure, which measures whether the completed APIs exactly match the ground-truth APIs.

In our study, we want to answer the following four research questions (RQs).

RQ1: How effective is our proposed approach APICom when compared with state-of-the-art baselines?

Results. In our study, we first consider the classical API recommendation approaches (i,e., BIKER [13], RACK [32], and CLEAR [40]) as the baselines. Moreover, since we are the first to model this problem as the automatic completion task, we also consider pre-trained models (i,e., CodeBERT [8], UniXcoder [11] and PLBART [2]) as the baselines. Our experimental results show that APICom outperforms both types of baselines in recommending the APIs that developers require.

RQ2: How effective is our proposed component ATCom in our proposed approach APICom?

Results. To show the effectiveness of ATCom, we consider three different classical adversarial training methods for data augmentation (i.e., FGSM [9], FGM [28], PGD [26]). Based on our ablation results, we find using ATCom can help to achieve the best performance for APICom.

RQ3: How does different pre-trained models affect the performance of APICom?

Results. To show the effectiveness of using CodeT5 in APICom, we consider three different pre-trained models (i.e., PLBART [2], UniXcoder [11], CodeBERT [8]). Based on our ablation results, we find using CodeT5 [38] can help to achieve the best performance for APICom.

RQ4: Whether using prompts can help to improve the performance of APICom?

Results. We conduct research to investigate the impact of using prompts on the performance of APICom, and our results show that APICom with prefixes can help to achieve the best performance.

Compare to previous studies on API recommendation [13, 32, 40], our study investigates a more practical problem. By providing some prompts with the query, our proposed approach APICom can help to provide accurate API completion suggestions, which can eventually generate APIs developers require.

To our best knowledge, the main contributions of our study can be summarized as follows.

- **Direction.** We are the first to study the task of API completion, which opens a new direction for automatic API recommendation.
- **Approach.** To solve this task, we propose a novel approach APICom based on natural language queries from developers and prefixes of APIs. APICom adopts the prompt learning paradigm and adversarial training-based data augmentation, and leverages the pre-trained CodeT5 model for learning API completion patterns automatically.
- **Evaluation.** We use a corpus containing 33k API queries and corresponding APIs as our experimental subjects. The extensive and comprehensive empirical study shows the competitiveness of our proposed approach APICom and the component setting rationality (such as the prompts, the adversarial training method, and the pre-trained model) in APICom.

To promote the replication of our research and encourage more follow-up studies in this direction, we share our corpus and scripts on our project homepage[2].

The rest of this paper is organized as follows. Section 2 provides research background (such as API recommendation and data augmentation technology) and illustrates our research motivation. Section 3 describes the framework and details of our proposed approach APICom. Section 4 shows empirical settings. Section 5 presents our analysis for different research questions. Section 6 discusses related studies to our work and shows the novelty of our study. Finally, Section 7 summarizes our work and shows potential future directions.

2 BACKGROUND AND RESEARCH MOTIVATION

In this section, we first introduce the background to the task of API recommendation and data augmentation. After analyzing these related studies, we emphasize the research motivation of our study.

2.1 API Recommendation

API recommendation task refers to recommending APIs suitable for developers based on their needs and historical behavior. The main purpose of the API recommendation task is to help developers use and integrate APIs more efficiently and improve the efficiency of their software development and maintenance. Gu et al. [10] proposed DeepAPI, which can generate API usage sequences for a given query. Rahman et al. [32] proposed RACK, which recommends relevant APIs for a query by exploiting keyword-API associations from the crowdsourced knowledge of Stack Overflow. Huang et al. [13] proposed BIKER. Specifically, BIKER first uses the word embedding technique to calculate the similarity between two queries. Then BIKER leverages Stack Overflow to extract candidate APIs, and ranks candidate APIs by considering the query's similarity with the information from both Stack Overflow posts and API documentation. Wei et al. [40] proposed CLEAR. this approach first embeds the whole sentence of queries and Stack Overflow posts with a BERT-based model. Then CLEAR uses contrastive learning for learning precise semantic representations of programming terminologies. Finally, CLEAR trains a re-ranking model to optimize its recommendation results

2.2 Data Augmentation

Data augmentation has been widely used in the NLP field, including rule-based methods and gradient-based methods.

For rule-based methods, researchers augmented the training data by using heuristic rules. For example, Wei et al. [39] proposed the EDA method, which consists of four simple but effective operations (such as synonym substitution, random insertion, random exchange, and random deletion). Xie et al. [42] proposed an unsupervised data augmentation method UDA. This method uses non-core word replacement technology, which replaces a certain proportion of unimportant words in the text with unimportant words in the dictionary.

For gradient-based methods (e.g., adversarial training), researchers augmented the training data by generating adversarial examples. The general principle of adversarial training can be summarized as the following maximum-minimum formula:

$$\underset{\theta}{min}\mathbb{E}_{(x,y)} \sim D\left[\underset{\|\delta\|\leq\epsilon}{max} L(f_\theta(X+\delta),y)\right] \quad (1)$$

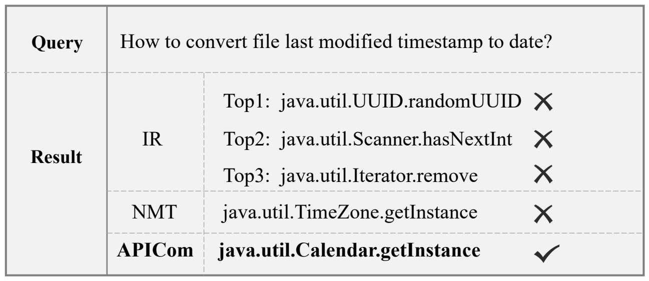
where x represents input, δ represents perturbation, y represents the label of the example, $max(L)$ represents the optimization objective, D is the training set, and $\mathbb{E}$ is the maximum likelihood estimation. For this kind of method, researchers mainly utilize classical adversarial training methods (such as FGSM [9], FGM [28], PGD [26]) for data augmentation, and the effectiveness of adversarial training in the NLP field has been widely verified [17, 19, 27, 29, 48].

2.3 Research Motivation

In this subsection, we use an example in Figure 2 to show the motivation of our study. In this figure, the query is "How to convert file last modified timestamp to date?". If we use the IR-based approach BIKER [13], the recommended top-3 APIs are "java.util.TimeZone.getInstance", "java.util.Scanner.hasNextInt", and "java.util.Iterator.remove". After a manual examination, we find the required API does not exist in these top-3 APIs. If we model this task as the automatic generation problem motivated by neural machine translation (NMT) and use the pre-trained model CodeBERT [8], the generated API is "javac.util.TimeZone.getInstance". Similar to the examples in Figure 1, there is an error with the third prefix. However, if the developer can provide the prefix of the required API (i.e., java.util.Calendar), our proposed approach APICom can generate the correct API. Based on this motivation example, we can find that API completion has great potential for effective API generation. Therefore, combined with the prompt learning paradigm, we propose an API completion approach APICom. Moreover, the quality of the dataset is vital for the effectiveness of APICom. In this study, we resort to the adversarial training method for performing data augmentation and design an effective method ATCom. ATCom can augment the training set by generating K adversarial

[2]https://anonymous.4open.science/r/APICom

examples for each original input. The novelty of ATCom is that this method can improve the stability of the normalization process while retaining the advantages of L_1 and L_2 normalization.



Query	How to convert file last modified timestamp to date?		
Result	IR	Top1: java.util.UUID.randomUUID	✗
		Top2: java.util.Scanner.hasNextInt	✗
		Top3: java.util.Iterator.remove	✗
	NMT	java.util.TimeZone.getInstance	✗
	APICom	**java.util.Calendar.getInstance**	✓

Figure 2: An example to show the motivation of our study

3 OUR PROPOSED APPROACH

We show the framework of our proposed approach APICom in Figure 3. Our approach consists of three parts: data pre-processing part, model architecture part, and model application part. In the rest of this section, we present detailed information for each part.

3.1 Data Pre-processing Part

To better simulate the scenario of API completion, we first use random masking to construct different prompts. Then we add a $< mask >$ tag after the incomplete API to indicate the masked part of the API, which can be predicted by the model trained by APICom.

Prompt learning [18, 35] can make pre-trained models bring the downstream task closer to the upstream task according to the specified prompt templates. According to the position of slots, templates can be divided into $Prefix$ mode and $Cloze$ mode. The former is that the unknown slot is at the beginning of the template, and the latter is that the unknown slot is at the uncertain position of the template. Researchers may construct different templates for the same task, and the choice of template plays an important role in the task of the prompt. Even the difference in a word can lead to a difference of more than 10 points in performance [25]. In our study, we generate different prompts through $Prefix$ mode.

To better train the API completion model, we need to construct different prompts. Assuming that the API t_i contains n words (separated by dots), we can define the masking operator as follows. Specifically, this operator first generates a random number n_{rand} ($1<n_{rand}<n$). Then the operator generates an incomplete API (i.e., a prefix prompt) with the last n_{rand} words masked. Examples of generating incomplete APIs are shown in the data pre-processing part of Figure 3.

The rationale for our designed masking operator can be summarized as follows. Firstly, since developers typically compose APIs in a linear manner, starting from the first word and moving toward the last word, our masking operator selects the last consecutive words to be masked. Secondly, our study mainly focuses on the API completion task, which requires incomplete APIs to have at least one prefix. Therefore, the masking operator can mask at most $n-1$ words. To generate incomplete APIs, we apply the masking operator with three different values of n_{rand}, which can result in three different incomplete APIs for each API. Although generating more incomplete APIs could potentially improve the performance of the trained model, it would also substantially increase the training data size and eventually increase the model construction cost.

Then, we take the generated prompt and the original query as the input for our proposed model. Therefore, the considered input can be expressed as $X = prompt \oplus X_{desc}$.

3.2 Model Architecture Part

In this part, ATCom automatically generates k adversarial examples according to the model input in the first phase. Then the generated adversarial examples and the original inputs are used as the training data to fine-tune our proposed API completion model in the second phase.

3.2.1 Data Augmentation Phase.

3.2.1 Data Augmentation Phase. Data augmentation can generate more training data by performing a series of transformations and extensions on the original corpus. Specifically, our proposed novel ATCom can generate k adversarial examples by adding perturbation to the embedding matrix for data augmentation.

Word Embedding. Word embedding can be used to capture the relationship between tokens by mapping text to the vector representation. For the given input X, APICom tokenizes this input by the BPE algorithm [43] to obtain the sequence $x = (x_1, \cdots, x_N)$, where N is the length of the sequence. To unify the length of the input sequence, we use the fill or truncation operation. In particular, we assume the maximum input length is n. Then if the input sequence size is less than n, we complete it with 0. Otherwise, if the input sequence size is more than n, we truncate the excess parts. After the above process, we can guarantee that the output of the word embedding layer is consistent $\boldsymbol{x} = (\boldsymbol{x_1}, \cdots, \boldsymbol{x_n})$.

ATCom. Like most adversarial training methods used in the NLP (natural language processing) domain [9, 14, 26, 28, 52], we also add perturbations to the embedding layer. Specifically, for a given input embedding sequence $\boldsymbol{x}$, ATCom aims to add perturbation δ and generate K adversarial examples $\left\{\boldsymbol{x^i_{adv}}\right\}_{i=1}^{K}$. Then CodeT5 utilizes the augmented training set (i.e., original input $\boldsymbol{x}$ and adversarial examples $\left\{\boldsymbol{x^i_{adv}}\right\}_{i=1}^{K}$) to fine-tune the model.

Like Formula (1), the purpose of our designed method ATCom is to find the most suitable perturbation. Different from previous studies [9, 28], ATCom does not directly calculate perturbations based on the parameter ϵ, but iterates multiple times to find the optimal perturbation. The input gradient g_t of each step is calculated as follows:

$$g_t = \nabla_{x_t} L(f_\theta(x_t), y) \tag{2}$$

where θ denotes the model parameter value, x denotes the model input, y denotes the label, and $L()$ denotes the loss function for model training. Then the perturbation of each step δ_{t+1} can be computed as follows:

$$\delta_{t+1} = \epsilon(g_t \cdot \|g_t\| / |g_t|) \tag{3}$$

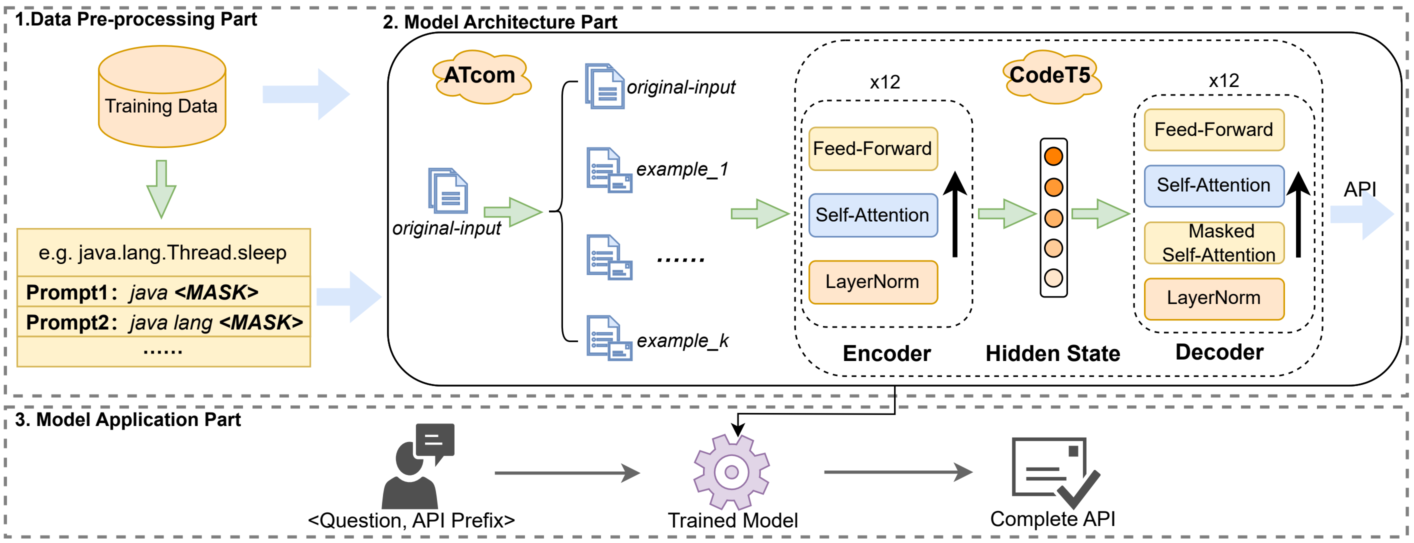


Figure 3: Framework of our proposed approach APICom

where ϵ is the constraint of perturbation (i.e., $L2$ norm of the distance between the original example and the adversarial example is always ϵ). Finally, ATCom takes the average gradient g_{avg} of K times of iteration when updating parameters' values, where

$$g_{avg} = (g_1 + \cdots + g_t)/t \tag{4}$$

Simon et al. [36] found that the adversarial loss can be expressed as a special regularization term as follows:

$$\widetilde{L}(x, y) \approx L(x, y) + \frac{\epsilon}{2} \|\partial_x L\|_q \tag{5}$$

where $\widetilde{L}$ represents the confrontation loss, L represents the original loss of the model, and $\frac{\epsilon}{2} \|\partial_x L\|_q$ represents the special regularization term. Based on their study, we aim to use $L1$ and $L2$ regularization methods to optimize the adversarial loss. The advantage of ATCom is that we use L_1 normalization to reduce the effect of large values on the vectors, and then apply L_2 normalization to ensure that the resulted vectors have a consistent length and can be summed to 1. Therefore, ATCom can improve the stability of the normalization process while retaining the advantages of L_1 and L_2 normalization.

3.2.2 *Model Fine-tuning Phase with CodeT5.*

CodeT5 [38] is a unified pre-trained encoder-decoder Transformer model that better leverages the code semantics conveyed from the developer-assigned identifiers. In this work, we adapt CodeT5 as our backbone model to solve the API completion task.

Encoder. The encoder module includes 12 sub-blocks and each block is made up of two subcomponents (i.e., a self-attention layer and a small feed-forward network). Specifically, Self-attention [6] is calculated by using queries (Q), keys (K) with the dimension d_k, and values (V) with the dimension d_v. In particular, the dot product of the queries and keys is first computed. Then the weight of each value is then calculated using the softmax function after each has been divided by $\sqrt{d_k}$. The output matrix can be calculated as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{6}$$

The feed-forward network (FFN) is composed of two linear transformations, which are separated by a ReLU activation and can be computed as follows:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{7}$$

Layer normalization [3] is applied to the input of each subcomponent. Following the layer normalization, a residual skip connection [12] adds each subcomponent's input to its output. The output of the encoder is defined as a hidden state vector $\boldsymbol{X}_{\text{out}}$.

Decoder. The decoder is similar to the encoder in structure except that it includes a masked self-attention layer, the purpose of which is to prevent the model from noticing unknown information during model training. In the decoding step, the model can generate the following API tokens by the encoder output $\boldsymbol{X}_{\text{out}}$. The network can predict the probability of the next token through the softmax layer, which can be defined as follows.

$$P(y_{t+1} \mid y_1, \ldots, y_t) = softmax(W \cdot X_{\text{out}} + b) \tag{8}$$

where y denotes the predicted token. We train our model parameters θ by the loss function L for a given input text x based on cross-entropy, which is defined as follows.

$$L = -\sum_{i=1}^{|y|} logP_\theta(y_i \mid y < i, x) \tag{9}$$

Since the output of our proposed approach APICom is an API, which can be treated as a token sequence, we find utilizing beam search [37, 41] can improve the performance. Beam search can return a list of the most likely output sequences, which can provide the developer with a few of the most likely API completion suggestions. Specifically, it uses the lowest cost k tokens of each time step by scanning the API tokens of each step one by one, where k denotes the beam width. After pruning any remaining branches, it continues to choose potential tokens for the subsequent tokens until it encounters the end-of-sequence sign. Finally, our model can return k candidate APIs for each query. We rank the generated candidate APIs according to their average probabilities during the beam search procedure.

3.3 Model Application Part

In this part, given the query content and prompt information provided by developers, our trained model can recommend a complete API.

4 EXPERIMENTAL SETUP

4.1 Research Questions

RQ1: How effective is our proposed approach APICom when compared with state-of-the-art baselines?

Motivation. In this RQ, we want to evaluate the quality of APIs generated by APICom by comparing APICom with state-of-the-art baselines in terms of two types of automatic performance measures.

RQ2: How effective is our proposed component ATCom in our proposed approach APICom?

Motivation. In this RQ, we want to conduct ablation experiments by investigating the performance impact on APICom of different adversarial training methods for data augmentation (i.e., FGSM [9], FGM [28], PGD [26], and our designed ATCom).

RQ3: How does different pre-trained models affect the performance of APICom?

Motivation. In this RQ, we want to investigate the impact of different pre-trained models (i.e., PLBART [2], UniXcoder [11], CodeBERT [8], and our considered CodeT5 [38]) on the performance of APICom.

RQ4: Whether using prompts can help to improve the performance of APICom?

Motivation. In this RQ, we want to explore the impact of prompts on the performance of APICom. Investigating the role of prompts is essential to understand how to optimize the model's performance, and reduce the amount of training data.

4.2 Experimental Subjects

To show the effectiveness of APICom, we select 33k high-quality query and API pairs from BIKER [13] as our experimental subject.

The statistical information of the corpus is shown in Table 2. In this table, we can find that the length of most queries and the corresponding APIs are mainly around 11 words and 4 words in the corpus after using the BPE algorithm. Moreover, 99.9% of queries are less than 48 words, and 99.9% of APIs are less than 16 words. In our empirical study, we utilize a random sampling method to split the corpus into a training set, a validation set, and a test set in the ratio of 80%:10%:10% by following previous studies [24, 46].

Table 2: Length statistics of our experimental subject

Query length statistics					
Average	Mode	Median	<16	<32	<48
11.086	9	10	84.2%	99.8%	99.9%
API length statistics					
Average	Mode	Median	<8	<12	<16
4.713	4	4	94.9%	99.8%	99.9%

4.3 Evaluation Measures

To compare the performance of APICom and the baselines, we first consider two performance measures: Mean Reciprocal Rank (MRR) [31] and Mean Average Rank (MAR) [34]. These two measures have also been widely used in previous API recommendation studies [13, 32, 40].

MRR. MRR [31] measures the model's ability to rank multiple candidate answers by calculating the average of the reciprocal of the best rank for each query, which is defined as follows:

$$MRR = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i} \tag{10}$$

where Q represents the total number of questions in the evaluation set, and $rank_i$ represents the position of the first correct answer in the answer set predicted for the $i\text{-}th$ question.

MAP. MAP [34] measures the average precision score of the Precision-Recall curve, which represents the trade-off between precision and recall at different thresholds. Therefore, the MAP can be defined as follows:

$$MAP = \frac{1}{|Q|}\sum_{i=1}^{|Q|} AveP\left(C_i, A_i\right) \tag{11}$$

where Q refers to the number of queries and $AveP(C_i, A_i)$ refers to the average accuracy.

Moreover, since we model the API recommendation problem as the automatic completion task, we further consider the Exact Match (EM) measure [7, 15], which measures whether the API generated by the model matches the ground-truth API. Notice EM@k indicates the correct rate when comparing the top n recommendations by using beam search. Since it is useless to measure the text similarity in the API recommendation task, some popular evaluation measures (such as BLEU [30] and ROUGE [33]) used in the text generation tasks are not considered in our study.

For these two types of measures, the higher the score, the better performance of the corresponding approach. To avoid result differences due to different implementations for performance metrics, we use the implementations provided by the nlg-eval library[3] for these performance measures, which can mitigate the threat to the internal validity of our empirical study.

4.4 Baselines

To demonstrate the competitiveness of our proposed approach APICom, we first compared APICom with state-of-the-art API

[3]https://github.com/Maluuba/nlg-eval

recommendation baselines, such as BIKER [13], RACK [32], and CLEAR [40]. We introduce the characteristics of these baselines as follows.

- **BIKER.** BIKER [13] uses word embedding technology to bridge the vocabulary gap and uses API to supplement the information to bridge the knowledge gap.
- **RACK.** RACK [32] uses keywords and API associations in stack overflow crowd-sourcing knowledge to recommend a list of related APIs for natural language queries.
- **CLEAR.** CLEAR [40] uses a BERT-based model to embed whole sentences of queries and stack overflow posts, and uses contrastive learning to train the model.

Since we model the API recommendation problem as the automatic completion task, we also compare APICom with baselines based on the pre-trained language models, such as CodeBERT [8], UniXcoder [11], and PLBART [2]. We introduce the details of these pre-trained language models as follows.

- **CodeBERT.** CodeBERT [8] trains the model based on the neural architecture of Transformer and uses the mixed objective function.
- **UniXcoder.** UniXcoder [11] is a unified cross-modal pre-trained model, which uses a masked attention matrix to control the model and uses cross-modal content to enhance code representation.
- **PLBART.** PLBART [2] is a text-to-text model, which learns program syntax, style, and logical flow that are crucial to program semantics.

For the first type of baseline [13, 32, 40], we implemented them according to the approach description, and the results of our implementations are very close to the results reported in these original studies. For the second type of baseline [2, 8, 11], we directly use the script shared by these original studies. To ensure a fair comparison between APICom and these baselines, we optimized the hyper-parameters in these approaches.

4.5 Implementation Details

Our proposed approach and some baselines were implemented based on the PyTorch framework. We use the packages Transformers[4] and HuggingFace [5] to implement APICom. To alleviate the overfitting issue, we employ an early stopping strategy that terminates training when the validation loss does not decrease for five consecutive iterations and saves the model's parameters with the best performance. The detailed hyper-parameter setting is shown in Table 3. For the hyper-parameter of baselines, we adopt the value setting of original papers.

Our experiments were conducted on a computer with an Intel (R) Core(TM) i5-13600K and a GeForce RTX4090 GPU with 24 GB memory.

[4]https://github.com/huggingface/transformers
[5]https://huggingface.co

Table 3: Hyper-parameters setting of our proposed approach APICom

Component	Hyper-parameter	Value
ATCom	K	4
	α	0.3
CodeT5	decoder_layers	12
	hidden_size	768
	max_input_length	48
	max_output_length	16
	beam_search_size	10

Table 4: Comparison results between APICom and API recommendation baselines

Approach Name	MRR	MAP	Approach Name	MRR	MAP
RACK	0.278	0.249	BIKER	0.586	0.613
CLEAR	0.591	0.602	**APICom**	**0.669**	**0.713**

5 EXPERIMENTAL RESULTS

5.1 RQ1: Comparision with baselines

In this RQ, we aim to investigate the competitiveness of our proposed approach APICom. Specifically, we compare our proposed approach with state-of-the-art API recommendation baselines [13, 32, 40] and baselines based on the pre-trained language models [2, 8, 11].

We first compare APICom and API recommendation baselines in terms of two automatic evaluation measures (i.e., MRR and MAR). The comparison results are shown in Table 4 and we emphasize the best result in bold. According to the experimental results, we find our proposed approach APICom can at least outperform the API recommendation baselines (i.e., CLEAR) by 13.20% in terms of MRR and APICom can at least outperform the API recommendation baselines (i.e., BIKER) by 16.31% in terms of MAP. This result indicates that our proposed approach APICom can show a significant improvement over traditional IR-based API recommendation baselines. Since APIs obtained by the IR recommendation method may not be what we need, APICom can generate APIs that are more in line with the actual needs by using prefix prompts in the model.

We second compare APICom and generation baselines based on the pre-trained language models in terms of EM performance measure. The comparison results are shown in Table 5, and we also emphasize the best result in bold. According to the experimental results, we find APICom can at least outperform this type of baselines by 40.02%, 32.70%, 29.52%, 23.85%, and 24.50% for EM@1, EM@2, EM@3, EM@4, and EM@5, respectively. This result shows the potential of APICom in improving the accuracy of API generation in software development. Specifically, compared to previous baselines based on the pre-trained language models, our proposed approach can help to improve semantic diversity through data augmentation, resulting in APIs with richer semantic information. Moreover, we can better align with user intent by utilizing prompt-based learning, resulting in APIs that are closer to their needs.

Table 5: Comparison results between APICom and baselines based on the pre-trained language models

Approach Name	EM@1	EM@2	EM@3	EM@4	EM@5
PLBART	24.30	33.90	40.50	44.70	47.40
UniXcoder	33.00	45.60	52.00	55.70	57.70
CodeBERT	40.10	49.70	54.70	58.90	60.00
APICom	**56.15**	**65.95**	**70.85**	**72.95**	**74.70**

Summary to RQ1: APICom can outperform state-of-the-art API recommendation baselines and pre-trained language models in terms of MRR, MAP, and EM performance measures.

5.2 RQ2: Ablation study on adversarial training approaches

In this RQ, we aim to investigate the performance impact on APICom of different adversarial training methods (i.e., FGSM [9], FGM [28], PGD [26], and our designed ATCom) and show the competitiveness of ATCom. Specifically, we use $w/o\ AT$ to denote the control approach, which does not consider the adversarial training method for APICom. In a similar way, we use FGSM (or FGM, PDG, and ATCom) to denote the control approach, which considers the corresponding adversarial training method for APICom. To guarantee a fair comparison, APICom and other control approaches follow the same experimental setup. We show the details of these adversarial training methods as follows.

- **FGSM.** FGSM [9] is an adversarial training method proposed by Goodfellow et al. [9]. The input gradient is $g = \nabla_x L(f_\theta(x), y)$, where θ denotes the model parameter value, x denotes the model input, y denotes the label, and $L()$ denotes the loss function of the training model. The perturbation goes to the maximum of the loss function along the gradient direction, which is expressed as $\delta = \epsilon sign(g)$, where $sign()$ is the regularization method, and ϵ is the constraint that the perturbation is limited by infinite norm (i.e., $\|\delta\|_\infty < \epsilon$).
- **FGM.** FGM [28] was proposed by Goodfellow et al. [28]. Unlike FGSM, which takes the *sign* function to regularize the gradient, FGM carries out $L2$ regularization on the gradient. In addition, FGSM takes the same step in each direction, while FGM scales according to the specific gradient to get better adversarial examples. The perturbation is expressed as $\delta = \epsilon(g/\|g\|_2)$, where ϵ is the constraint of perturbation ($L2$ norm of the distance between the original example and the adversarial example is always ϵ).
- **PDG.** FGM calculates the perturbation directly through the parameter ϵ, which may not be optimal. Therefore, PGD [26] was improved and iterated several times to find the optimal perturbation. The input gradient of each step is expressed as $g_t = \nabla_{x_t} L(f_\theta(x_t), y)$ and the perturbation of each step is expressed as $\delta_t = \epsilon(g_t/\|g_t\|_2)$. In the iteration, δ_t is gradually accumulated, and only the gradient calculated by the last $x_t + \delta_t$ is used when the parameters are finally updated.

Table 6: Ablation study results between ATCom and other adversarial training methods

AT Method	EM@1	EM@2	EM@3	EM@4	EM@5
w/o AT	52.60	61.65	66.30	68.75	70.60
with FGSM	53.39	62.31	66.90	69.15	70.75
with FGM	54.85	64.00	67.65	70.55	72.15
with PDG	55.15	64.70	68.70	70.70	72.40
with ATCom	**56.15**	**65.95**	**70.85**	**72.95**	**74.70**

In Table 6, we show the ablation study results between ATCom and other classical adversarial training methods in terms of EM performance measure and mark the best result in bold.

Firstly, we find that APICom with classical adversarial training (AT) outperformed APICom without AT in all cases. Specifically, when using FGSM, FGM, and PGD methods, the performance of APICom with AT can be Significantly improved by at least 1.50%, 4.28%, and 4.85% in terms of EM@1 measure. This shows that the use of Adversarial Training Augmentation can effectively enhance the quality of the generated APIs in the API completion task.

Secondly, we find that APICom with our designed method ATCom can outperform the other classical AT methods. Specifically, the performance of APICom with ATCom can be at least improved by 5.17%, 2.37%, and 1.81% when compared to APICom with FGSM, FGM, and PGD. The comparison results show that ATCom can improve the performance while ensuring stability by using both $L1$ and $L2$ regularization methods to optimize the adversarial loss. Therefore, our proposed new adversarial training method ATCom can effectively improve the overall performance of APICom.

Summary to RQ2: According to the results of our ablation experiment, we find using our designed adversarial training method ATCom can help to achieve the best performance for APICom.

5.3 RQ3: Ablation study on pre-trained models

In this RQ, we aim to demonstrate the effectiveness of using CodeT5 for APICom. To answer this RQ, we conducted a comparative analysis with the other three classical pre-trained models discussed in Section 4.4. To ensure a fair comparison, we used the same experimental setup in the model fine-tuning phase (details can be found in Section 3.2.2) for these pre-trained models.

We use Table 7 to show the performance impact of different pre-trained models and highlight the best performance in bold. The comparison results show that APICom with CodeT5 can outperform the other three classical pre-trained models. Specifically, compared to the other pre-trained models, CodeT5 can at least improve the performance by 19.09%, 13.22%, 10.10%, 8.40%, and 7.48% in terms of EM@1, EM@2, EM@3, EM@4, and EM@5, respectively. These results imply that CodeT5 is better suited for the task of API complementation than the other pre-trained models.

Table 7: Ablation study results between CodeT5 and other pre-trained language models

Approach Name	EM@1	EM@2	EM@3	EM@4	EM@5
PLBART	40.30	48.80	54.50	57.20	60.40
UniXcoder	38.90	49.30	54.80	59.10	61.55
CodeBERT	47.15	58.25	64.35	67.30	69.50
CodeT5	**56.15**	**65.95**	**70.85**	**72.95**	**74.70**

Table 8: Comparison results between whether using prompt

Approach Name	EM@1	EM@2	EM@3	EM@4	EM@5
w/o prompt	49.40	59.30	63.10	65.10	67.50
with prompt	**56.15**	**65.95**	**70.85**	**72.95**	**74.70**

Summary to RQ3: According to the results of our ablation experiment, we find using the recently proposed pre-trained model CodeT5 can help to achieve the best performance for APICom.

5.4 RQ4: Ablation study on using prompts

In this RQ, we aim to investigate the impact of using prompts on the performance of APICom. Specifically, we compare the performance of models trained by APICom with and without prompts. To ensure a fair comparison, we follow the same experimental setup for these two approaches.

We show the comparison results in Table 8. Specifically, APICom with prompts can improve the performance by 13.66%, 11.21%, 12.28%, 12.06%, 10.67% in terms of EM@1, Em@2, EM@3, EM@4, EM@5 when compared with APICom without prompts. These results show that incorporating prompts can provide valuable context information for the API completion model, which can eventually result in more accurate suggestions.

Summary to RQ4: According to the results of our ablation experiment, we find using prompts can help to achieve the best performance for APICom.

6 DISCUSSION

6.1 Hyper-parameter Analysis

In this subsection, we perform a sensitivity analysis on the hyper-parameters of APICom to explore their optimal settings. Here we mainly focus on the hyper-parameter K, which is the number of adversarial examples. Figure 4 shows the results of the sensitivity analysis. Except for the hyper-parameter of the current analysis, the values of all hyper-parameters are set to our optimal settings. In this figure, the left vertical axis is used for measuring MAP and MRR, and the right vertical axis is used for measuring EM@1. Based on the sensitivity analysis results, we can find the optimal value for the hyper-parameter K is 4 for our investigated API completion task.

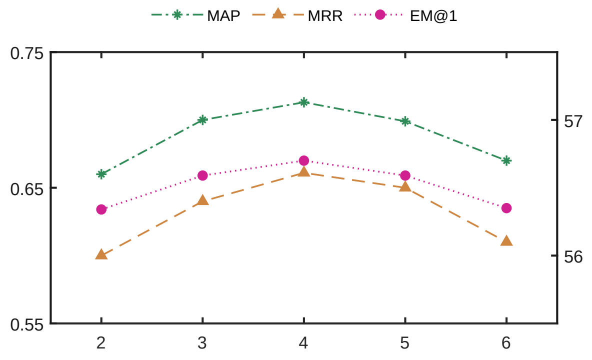


Figure 4: Sensitivity Analysis on the number of adversarial examples (i.e., the hyper-parameter K)

Table 9: The performance of APICom with different prefix lengths

Prefix Length	EM@1	EM@2	EM@3	EM@4	EM@5
0 prefix	49.40	59.30	63.10	65.10	67.50
1 prefix	51.30	61.20	64.40	66.60	68.10
2 prefixes	**56.00**	**66.40**	**70.80**	**72.90**	**74.00**

6.2 Prefix Length Influence of APICom

In this subsection, we want to analyze how the lengths of the prefixes influence the performance of APICom. We conduct a sensitivity study on the performance impact of prompts with different prefix lengths (i.e., 0 prefix, 1 prefix, 2 prefixes). Specifically, 0 prefix stands for no prompt. 1 prefix (or 2 prefixes) stands for only providing the first one prefix (or the first two prefixes). The remaining parts follow the same experimental setup to guarantee a fair comparison.

The final results are shown in Table 9. The average accuracy of APICom with 2 prefixes was 74.00%, which is significantly higher than that of APICom without prompts (67.50%) and with 1 prefix (68.10%) in terms of EM@5. Therefore, increasing the prefix length (i.e., providing more context information) can lead to better API completion suggestions.

6.3 Performance Improvement by API Correctness Check

In this subsection, we aim to further improve the performance of APICom by considering API correctness check. Specifically, these APIs are submitted to an external API library to determine whether they have the correct syntax and semantics. The external API library contains all available APIs, which serve as a reference for syntax and semantic constraints. If the generated API cannot be found in the library, it is considered incorrect. For all APIs generated by beam search, we check them in sequence until we find five APIs, which exist in the external API library.

Table 10 shows the final comparison results. In this table, we also concern EM measure and mark the best one of each measure in bold. The experimental results show that by using APICheck,

Table 10: Comparison results of whether to use the module APICheck or not

Approach Name	EM@1	EM@2	EM@3	EM@4	EM@5
w/o APICheck	56.15	65.95	70.85	72.95	74.70
with APICheck	**59.40**	**67.85**	**71.55**	**74.15**	**76.25**

APICom can further improve the model performance. Specifically APICom can improve the performance by 5.79%, 2.88%, 0.99%, 1.64%, and 2.07% in terms of EM@1, EM@3, EM@5, MRR, and MAP, respectively.

6.4 Limitations of APICom

After automatic evaluation, we can find that APICom can achieve better performance than baselines. However, we also notice that APICom may generate low-quality APIs when compared to ground truth. In this subsection, we identify three challenge types and analyze them in details.

The first challenge type is that developers may not accurately describe their queries, which may result in poor-quality generated APIs. For example, developers want to find what is the usage of Interrupts in Java, The ground truth is "java. lang. Thread. stop", but the developers incorrectly describe it as "how Java is partitioned", resulting in the generation of incorrect APIs. One possible solution is to use query reformulation methods [4] to alleviate this issue.

The second challenge type arises when the required API consists of a large number of tokens after using the BPE algorithm. In such cases, even if the developer correctly enters the prefix, there is still a possibility of generating the wrong API. For example, the developer wants to create HTML documents from HTML strings in Java, and the corresponding ground truth should be "javax.swing.text.html.HTMLDocument.setOuterHTML". Even though the prefix given by the developer is "Javax.swing", the API generated by APICom is "javax.swing.jeditorpane.setpage", which is different from the ground truth.

The third challenge type is related to the trained model. This shows there still exists performance improvement room for the current model. Therefore, further augmenting the training data and improving the diversity of the gathered queries and APIs may alleviate this issue.

6.5 Threats to Validity

Internal threats. The first internal threat is the potential errors during the implementation of APICom and the baselines. To mitigate this threat, we use mature frameworks (such as PyTorch and Transformer) to ensure the correctness of our code implementation. For the baseline, we use scripts shared in previous work to alleviate this threat. The second threat is related to our hyper-parameter settings. It may take a huge computational csot to find the optimal hyper-parameter settings for APICom. However, based on our current hyper-parameter settings, APICom can still achieve better performance than baselines.

External threats. The first external threat is related to the quality of our constructed corpus. To alleviate this threat, we use the experimental subject shared by Huang et al. [13], which can guarantee the generalization of our empirical findings. The second threat is we only use the dataset shared by Huang et al. [13] to demonstrate the effectiveness of APICom, which only supports Java programming language. Therefore, the performance of APICom may differ when performing API recommendations for other programming languages, especially some low-resource programming languages (such as Ruby, and Go).

Construct threats. The main construct threat is related to our used performance measures. To evaluate the effectiveness of APICom, we consider IR-based measures used by previous API recommendation studies [13, 32, 40]. Moreover, to evaluate our API completion method more accurately, we also consider EM metric [7, 15], which can measure whether the generated API can completely match the ground-truth API.

7 CONCLUSION

In this study, we provide a new direction for API recommendation by modeling this problem as the automatic completion task. In the completion task, the developer can input the query and the API prefixes (i.e., the prompt) they know. Then we propose a novel API completion approach APICom with prompt learning and adversarial training-based data augmentation. Specifically, APICom trains the API completion model through API prompts and adversarial examples generated by our designed method ATCom in the embedding layer. In our experimental studies, we evaluate the effectiveness of APICom in terms of two types of performance measures. Final results show that APICom can outperform state-of-the-art API recommendation baselines and baselines based on the pre-trained language models. Finally, we also verify the rationality of the component setting in APICom, such as our designed adversarial training method, the used pre-trained model, and prompt learning.

In the future, we first want to apply our proposed approach to other programming languages to verify the effectiveness of APICom. We second want to improve the performance of APICom by considering more advanced data augmentation techniques and large language models (such as Codex or ChatGPT). Finally, we want to automatically generate parameters that are needed for invoking APIs, which can further improve the practicability of APICom.

ACKNOWLEDGEMENT

Yafeng Gu and Yiheng Shen have contributed equally to this work and they are co-first authors. The authors would like to thank three anonymous reviewers for their insightful comments and suggestions, which can substantially improve the quality of this work. This work is supported in part by the Innovation Training Program for College Students (2023214, 2023356).

REFERENCES

[1] Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2020. A Transformer-based Approach for Source Code Summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 4998–5007.

[2] Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2655–2668.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[4] Kaibo Cao, Chunyang Chen, Sebastian Baltes, Christoph Treude, and Xiang Chen. 2021. Automated query reformulation for efficient search based on query logs from stack overflow. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1273–1285.

[5] Songqiang Chen, Xiaoyuan Xie, Bangguo Yin, Yuanxiang Ji, Lin Chen, and Baowen Xu. 2020. Stay professional and efficient: automatically generate titles for your bug reports. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 385–397.

[6] Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long Short-Term Memory-Networks for Machine Reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 551–561.

[7] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555* (2020).

[8] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[9] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples. *3th International Conference on Learning Representations, ICLR 2015* (2015).

[10] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2016. Deep API learning. In *Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering*. 631–642.

[11] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified Cross-Modal Pre-training for Code Representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 7212–7225.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[13] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. 2018. API method recommendation without worrying about the task-API knowledge gap. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 293–304.

[14] Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2020. SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2177–2190.

[15] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*. 4171–4186.

[16] An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. 2015. Combining deep learning with information retrieval to localize buggy files for bug reports (n). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 476–481.

[17] Linyang Li and Xipeng Qiu. 2021. Token-aware virtual adversarial training in natural language understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 8410–8418.

[18] Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190* (2021).

[19] Zhongyang Li, Xiao Ding, and Ting Liu. 2018. Generating reasonable and diversified story ending using sequence to sequence model with adversarial training. In *Proceedings of the 27th International Conference on Computational Linguistics*. 1033–1043.

[20] Zheng Li, Yonghao Wu, Bin Peng, Xiang Chen, Zeyu Sun, Yong Liu, and Doyle Paul. 2022. Setransformer: A transformer-based code semantic parser for code comment generation. *IEEE Transactions on Reliability* (2022).

[21] Zheng Li, Yonghao Wu, Bin Peng, Xiang Chen, Zeyu Sun, Yong Liu, and Deli Yu. 2021. SeCNN: A semantic CNN parser for code comment generation. *Journal of Systems and Software* 181 (2021), 111036.

[22] Hao Lin, Xiang Chen, Xuejiao Chen, Zhanqi Cui, Yun Miao, Shan Zhou, Jianmin Wang, and Zhan Su. 2023. Gen-FL: Quality prediction-based filter for automated issue title generation. *Journal of Systems and Software* 195 (2023), 111513.

[23] Ke Liu, Guang Yang, Xiang Chen, and Chi Yu. 2022. Sotitle: A transformer-based post title generation approach for stack overflow. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 577–588.

[24] Shangqing Liu, Yu Chen, Xiaofei Xie, Jing Kai Siow, and Yang Liu. 2020. Retrieval-Augmented Generation for Code Summarization via Hybrid GNN. In *International Conference on Learning Representations*.

[25] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021. GPT understands, too. *arXiv preprint arXiv:2103.10385* (2021).

[26] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings* (2018).

[27] Xiaofeng Mao, Yuefeng Chen, Ranjie Duan, Yao Zhu, Gege Qi, Xiaodan Li, Rong Zhang, Hui Xue, et al. 2022. Enhance the visual representation via discrete adversarial training. *Advances in Neural Information Processing Systems* 35 (2022), 7520–7533.

[28] Takeru Miyato, Andrew M Dai, and Ian Goodfellow. 2017. Adversarial training methods for semi-supervised text classification. *5th International Conference on Learning Representations, ICLR 2017* (2017).

[29] John Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 119–126.

[30] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.

[31] Dragomir R Radev, Hong Qi, Harris Wu, and Weiguo Fan. 2002. Evaluating Web-based Question Answering Systems.. In *LREC*. Citeseer.

[32] Mohammad Masudur Rahman, Chanchal K Roy, and David Lo. 2016. Rack: Automatic api recommendation using crowdsourced knowledge. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, 349–359.

[33] Lin CY ROUGE. 2004. A package for automatic evaluation of summaries. In *Proceedings of Workshop on Text Summarization of ACL, Spain*.

[34] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. 2008. *Introduction to information retrieval*. Vol. 39. Cambridge University Press Cambridge.

[35] Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980* (2020).

[36] Carl-Johann Simon-Gabriel, Yann Ollivier, Leon Bottou, Bernhard Schölkopf, and David Lopez-Paz. 2019. First-order adversarial vulnerability of neural networks and input dimension. In *International conference on machine learning*. PMLR, 5809–5817.

[37] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* 27 (2014).

[38] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. 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*. 8696–8708.

[39] Jason Wei and Kai Zou. 2019. 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)*. 6382–6388.

[40] Moshi Wei, Nima Shiri Harzevili, Yuchao Huang, Junjie Wang, and Song Wang. 2022. Clear: contrastive learning for api recommendation. In *Proceedings of the 44th International Conference on Software Engineering*. 376–387.

[41] Sam Wiseman and Alexander M Rush. 2016. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960* (2016).

[42] Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. 2020. Unsupervised data augmentation for consistency training. *Advances in Neural Information Processing Systems* 33 (2020), 6256–6268.

[43] Tianci Xu and Peng Zhou. 2022. Feature Extraction for Payload Classification: A Byte Pair Encoding Algorithm. In *2022 IEEE 8th International Conference on Computer and Communications (ICCC)*. IEEE, 1–5.

[44] Guang Yang, Xiang Chen, Yanlin Zhou, and Chi Yu. 2022. Dualsc: Automatic generation and summarization of shellcode via transformer and dual learning. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 361–372.

[45] Guang Yang, Ke Liu, Xiang Chen, Yanlin Zhou, Chi Yu, and Hao Lin. 2022. CCGIR: Information retrieval-based code comment generation method for smart contracts. *Knowledge-Based Systems* 237 (2022), 107858.

[46] Guang Yang, Yanlin Zhou, Xiang Chen, and Chi Yu. 2021. Fine-grained pseudo-code generation method via code feature extraction and transformer. In *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 213–222.

[47] Guang Yang, Yu Zhou, Xiang Chen, Xiangyu Zhang, Tingting Han, and Taolue Chen. 2023. ExploitGen: Template-augmented exploit code generation based on CodeBERT. *Journal of Systems and Software* 197 (2023), 111577.

[48] Jin Yong Yoo and Yanjun Qi. 2021. Towards Improving Adversarial Training of NLP Models. In *Findings of the Association for Computational Linguistics: EMNLP 2021*. 945–956.

[49] Chi Yu, Guang Yang, Xiang Chen, Ke Liu, and Yanlin Zhou. 2022. BashExplainer: Retrieval-Augmented Bash Code Comment Generation based on Fine-tuned CodeBERT. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 82–93.

[50] Motahareh Bahrami Zanjani, Huzefa Kagdi, and Christian Bird. 2015. Automatically recommending peer reviewers in modern code review. *IEEE Transactions on Software Engineering* 42, 6 (2015), 530–543.

[51] Yanlin Zhou, Shaoyu Yang, Xiang Chen, Zichen Zhang, and Jiahua Pei. 2023. QTC4SO: Automatic Question Title Completion for Stack Overflow. In *31st IEEE/ACM International Conference on Program Comprehension (ICPC)*.

[52] Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. 2020. Freelb: Enhanced adversarial training for natural language understanding. *8th International Conference on Learning Representations, ICLR 2020* (2020).