Evaluating prompt-learning-based API review classification through pre-trained models

Xia Li, Allen Kim

The Department of Software Engineering and Game Design and Development,
Kennesaw State University,
Marietta, USA

Abstract. To improve the work efficiency and code quality of modern software development, users always reuse Application Programming Interfaces (APIs) provided by third-party libraries and frameworks rather than implementing from scratch. However, due to time constraints in software development, API developers often refrain from providing detailed explanations or usage instructions for APIs, resulting in confusion for users. It is important to categorize API reviews into different groups for easily usage. In this paper, we conduct a comprehensive study to evaluate the effectiveness of prompt-based API review classification based on various pre-trained models such as BERT, RoBERTa, BERTOverflow. Our experimental results show that prompts with complete context can achieve best effectiveness and the model RoBERTa outperforms other two models due to the size of training corpus. We also utilize the widely-used fine-tuning approach LoRA to evaluate that the training overhead can be significantly reduced (e.g., 50% reduction) without the loss of the effectiveness of classification.

Keywords: Software engineering, API review classification, pre-trained models, fine-tuning

1 Introduction

Software systems have been widely used in almost all aspects of human life in recent decades, making our lives more and more convenient. To improve the work efficiency and code quality of modern software development, users always reuse Application Programming Interfaces (APIs) provided by third-party libraries and frameworks rather than implementing from scratch. For example, the Java Software Development Kit (Java SDK) provides many packages for users to conveniently reuse the APIs. Python also provides a lot of APIs to interact with Amazon Web Services (AWS), which is one of the most popular cloud service providers, to reduce both development time and effort. However, due to time constraints in software development, API developers often refrain from providing detailed explanations or usage instructions for APIs, resulting in confusion for users. Previous studies [2] have revealed that API documentation frequently faces significant quality issues, including incompleteness, outdated information, and inaccuracies. To find better and practical explanation of APIs, users typically turn to popular Q&A platforms like Stack

Overflow[1] to find API information that suits their needs. These platforms host reviews where developers discuss various aspects of APIs, such as usability, documentation, performance, etc. By identifying the specific focus or classification of a review, developers can more easily access the information that is most relevant to their requirements. This emphasizes the importance of developing automated techniques that can effectively categorize API reviews into different groups. To date, various machine learning techniques have been used to improve the performance of API review classification. Uddin et al. [3] was the first to categorize API reviews into different predefined aspects, such as usability or security. Uddin et al. also suggested a machine learning-based technique [4] to label Stack Overflow sentences with various categories. Lin et al. [5] developed a pattern-driven approach to classify API opinions from Q&A platforms into specific categories. Recently, pre-trained foundation models, such as BERT [15] and GPT [18], have gained significant performance across diverse AI domains, including natural language processing (NLP), computer vision (CV), and graph learning (GL). These models have been successfully applied to various downstream tasks, including text classification [16] and image classification [17]. In the field of API review classification, pre-trained models have also shown promising results. For example, Yang et al. [6] evaluated the aspect-based API review classification task by fine-tuning six pre-trained models and compared them on a widely-used API review benchmark. However, a study [19] demonstrates that there are gaps between general pre-trained models and specific classification tasks since pre-trained models is only used to generate an embedded tokens that are used for traditional downstream tasks. Thus, the power of pre-trained models is not fully utilized. To bridge this gap, certain prompts can be appended after the input sequence and the target task is masked so that pre-trained models can predict the masked label then [19]. Furthermore, the current survey on prompt engineering [21] shows that varying prompts can influence the performance of the pre-trained models, highlighting the importance of evaluating the effect of different prompt templates and the need to evaluate their influence on API review classification. In this paper, we conduct a comprehensive study to evaluate the effectiveness of prompt-based API review classification based on various pre-trained models such as BERT [15], RoBERTa [1], BERTOverflow [14]. We also utilize the widely-used fine-tuning approach LoRA to demenstrate that the training overhead can be significantly reduced (e.g., 50% reduction) without the loss of the effectiveness of classification.

The paper is organized as follows. In Section 2, we provide an overview of related studies on API review classification. In Section 3, we describe the methodology used in our study. Sections 4 and 5 present the experimental setup and analysis of the results, respectively. We address the potential threats to the validity of our findings in Section 6 and conclude the paper in Section 7.

[1] https://stackoverflow.com/

2 Related Work

In this section, we discuss some related studies of API review classification and pre-trained models for software engineering.

2.1 API review classification

Uddin et al. [3] proposed to categorize API reviews into different predefined aspects, such as usability or security. They also suggested a machine learning-based technique [4] to label Stack Overflow sentences with various categories. Lin et al. [5] developed a pattern-driven approach to classify API opinions from Q&A platforms into specific categories. Yang et al. [6] evaluated the aspect-based API review classification task by fine-tuning six pre-trained models and compared them on a widely-used API review benchmark. Zhang et al. [7] used sentiment analysis to extract problematic API features from online discussions. Ahasanuzzaman et al. [8] concentrated on detecting API-related posts on Stack Overflow and developed a supervised learning approach called CAPS based on five different dimensions and Conditional Random Field (CRF) technique. Treude et al. [9] employed machine learning techniques to identify insightful API-related sentences from Stack Overflow discussions and leveraged these findings to improve API documentation.

2.2 Pre-trained models for Software Engineering

Nowadays, pre-trained models have been widely used in various fields of software engineering. Hey et al. [22] proposed NoRBERT to fine-tune BERT model and apply it to different tasks for requirements classification. CodeBERT [10], a transformer-based model pre-trained on a large-scale corpus of code from GitHub, has been widely used for tasks like code summarization and completion. Zeng et al. [11] performed an extensive study of eight pre-trained models for program understanding and generation. Luo et al. [20] proposed PRCBERT for requirement classification using BERT model by applying prompt templates for accurate requirements classification. Xia et al. [12] performed extensive studies on directly applying pre-trained models for automated program repair.

3 Study Design and Approach

In this section, we introduce the general process of our study shown in the Figure 1. We describe the key approaches such as data preprocessing (Section 3.1), new input creation based on prompt templates (Section 3.2) and the LoRA fine-tuning approach(Section 3.3).

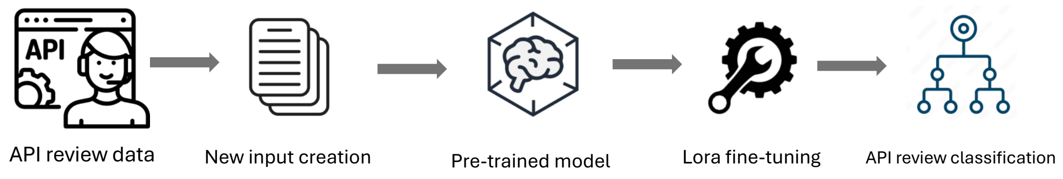

Fig. 1. Overall process of the study

3.1 Data preprocessing

In our study, we use the dataset created by Uddin et al. [3], including 4,522 sentences extracted from 1,338 Stack Overflow posts where users discussed and reviewed various APIs. The distribution of the dataset is shown in Table 1. Please note that each sentence is manually labeled with one or more API review categories. In our paper, we only use 4,307 sentences that are associated with a single API review category. We pre-process the dataset using common natural language processing techniques, including stemming, lemmatization, stop-word removal, and conversion to lowercase, utilizing the widely used NLTK toolkit[2]. We then generate various prompts based on the new dataset as the input of pre-trained models for API review classification.

Table 1. Distribution of API review categories

Category	Number of Instances	Category	Number of Instances
Performance	348	Usability	1,437
Security	163	Community	93
Bug	189	Compatibility	93
Portability	70	OnlySentiment	348
Documentation	256	Legal	50
Others	1,699		

3.2 Prompt engineering

Conventional classification tasks often rely on pre-trained models to extract a fixed vector representation of the input sequence. This representation is subsequently processed by an additional neural network, such as an RNN or CNN, to predict the various classes. However, this approach leads to a weak correlation between the input sequence and the final classification task. To address this issue, our study

[2] https://www.nltk.org/

adopts an alternative approach inspired by a study called Pattern-Exploiting Training (PET) [19]. In details, we perform the following two tasks by modifying original input sentences as the new input that is compatible with pre-trained models for API review classification. First, we insert a masked label as a special token into the original sentence to generate a new input. The masked label will be the target value for prediction during the training process. We use the cross-entropy loss function to fine-tune the parameters of pre-trained models based on the difference between the actual value and the predicted value of the masked label. Cross-entropy is chosen because API review classification is the classic multi-class classification problem. By embedding the target label prediction directly into the model's input, our approach strengthens the connection between the input sequence and the classification objective, enabling a more precise and effective fine-tuning process. Second, we design and evaluate various prompt templates for API review classification. The intuition is that pre-trained models (e.g., GPT, BERT) are highly sensitive to prompt variations based on recent work related with prompt engineering [21], which has been applied in the field of software engineering. For example, prior research [20] has demonstrated that the manually inserted prompts can significantly improve performance in the area of software requirements classification. However, the impact of different prompts on API review classification remains unexplored. To illustrate our approach, we use the API review sentence "You could even put the unexpected state your getter found in the exception message" as an example, which is categorized in the "Bug" category in the dataset. In details, we design and experiment with multiple prompt templates by inserting them at different positions in the original API review sentences. Figure 3.2 presents the templates used in our study. In these templates, [CLS] represents a special token used by the pre-trained models (e.g., BERT [15]) at the beginning of the input text, while [SEP] is a separator token that marks the boundary between different sentences. The masked token [M] corresponds to the API review category (e.g., performance, security, portability) that the pre-trained models predict. This approach allows us to systematically analyze how different prompt designs influence the model's classification performance.

3.3 Lora fine-tuning

Fine-tuning pre-trained language models (e.g., BERT, GPT-3, LLaMA) for specific tasks is essential to enhance their performance. However, updating all parameters of these models is often impractical due to their immense size (e.g., 175 billion parameters for GPT-3). To address this challenge, LoRA (Low-Rank Adaptation) offers a more efficient alternative by drastically reducing the number of trainable parameters. In our study, we investigate if LoRA can maintain the effectiveness of API review classification with fine-tuning all parameters while significantly reducing computational overhead.

Fig. 2. Prompt templates

Template 1: **[CLS]** You could even put the unexpected state your getter found in the exception message. **[SEP]** This API review is for **[M]**. **[SEP]**	**Template 2:** **[CLS]** Following sentence is an API review for **[M]**. **[SEP]** You could even put the unexpected state your getter found in the exception message**.[SEP]**
Template 3: **[CLS]** Given the following statement: "You could even put the unexpected state your getter found in the exception message"**[SEP]** Question: what type of API review is it? **[SEP]** Answer: **[M] [SEP]**	**Template 4:** **[CLS]** "You could even put the unexpected state your getter found in the exception message" is an API review related to **[M]. [SEP]**

LoRA works by freezing the pre-trained model's original weights while introducing two smaller matrices holding the updated weights through low-rank decomposition. This approach allows the model to adapt to input data while minimizing the number of parameters that need to be trained. Specifically, the original weight matrix W0 (with size $d \times d$) keeps unchanged while ΔW is the new updated weights with the same size $d \times d$ size. LoRA also introduces a new parameter r to reduce the size of the matrix ΔW to split into two smaller matrices, A and B, with size of $r \times d$, and $d \times r$. During training process with LoRA, only the weights of matrices A and B are updated. The updated weights can be merged with the weights of the base model. After the training process, the new input with size of $1 \times d$ will be multiplied by both W and ΔW, resulting in two output vectors with size d. The two vectors are then concatenated element-wise to get a final vector with size h.

4 Experimental Design

In this section, we introduce the experimental design of our study including sampling strategy (Section 4.1) and the experimental configuration (Section 4.2).

4.1 Sampling strategy

From the table 1 in section 3.1, we can find that the dataset is imbalanced, with varying numbers of instances across different categories. For example, the proportion of each API review category ranges from 1.1% (e.g., Portability) to 37.6%. We utilize the sampling strategy inspired by Uddin et al [13] who addressed this issue. Specifically, for each API review category, we consider all corresponding instances as positive samples, while the remaining instances from other categories are treated as negative samples.

4.2 Experimental configuration

In our study, we use three foundation pre-trained models BERT [15], RoBERTa [1], BERTOverflow [14] that can be downloaded from the popular AI community Hugging Face[3]. BERT and RoBERTa are pre-trained on natural languages while BERTOverflow is pre-trained on corpus extracted from the posts on Stack Overflow. For the hyperparameter, we set the maximum input sequence length as 256, batch size as 8, learning rate as $5e^{-5}$, epochs as 16. We use the popular evaluation metrics precision, recall and F1 score for classification problems as follows.

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Where TP indicates the number of True Positives, FP indicates the number of False Positives and FN is the number of False Negatives. We use AdamW optimizer [23] during the training process. We use 80% of the original dataset as training set and 20% of original dataset as test set. We apply the 10-fold cross-validation for each prompt introduced in Section 3.2. Please note that we apply such cross-validation for each category under the sampling strategy introduced in Section 4.1. The training and inference steps of the API review classification are executed on a computer with Intel Core 13900K CPU, 32GB memory and NVIDIA RTX 4090 GPU.

5 Results Analysis

In this paper, we investigate the following two research questions:

– **RQ1:** How is the effectiveness of API review classification based on various prompt templates?
– **RQ2:** How is the efficiency of API review classification by LoRA fine-tuning?

5.1 Effectiveness of API review classification based on various prompt templates

In this RQ, we investigate the performance of API review classification on the three pre-trained models: BERT, RoBERTa and BERTOverflow based on the four prompt designs in the Section 3.2. Table 2 shows the results based on the four

[3] Hugging Face. https://huggingface.co/

templates in terms of the evaluation metrics precision, recall and F1 score. Please note that we calculate the average values of the 10-fold cross-validation for each category. We finally get the average values of the metrics across 9 categories in the table 1. From the results, we have the following findings. First, we can find that the performance of template 3 is better than other templates. The possible reason is that template 3 provides a complete context ("question" and "answer") that can let pre-trained models thinking and reasoning. Second, the overall performance of BERTOverflow is worse than other two models for all metrics. For example, the F1 score of BERTOverflow for template 1 is 81.86% while the F1 score of RoBERTa is 83.31%. The F1 score of BERTOverflow is also less than BERT for all templates. The possible reason is that the size of training data of BERT (3.3 billion words) is much larger than the size of BERTOverflow (only 152M sentences). Similarly, RoBERTa is better than BERT due to the larger size of training corpus and dynamic masking during their training process [1]. This result shows that even we utilize useful prompt templates, the training corpus size of pre-trained models still plays the most important role for the classification task.

Table 2. Effectiveness of API review classification based on various prompt templates

Templates	Models	Avg Precision	Avg Recall	Avg F1 Score
Template 1	RoBERTa	82.92%	83.71%	83.31%
	BERT	81.53%	82.35%	81.94%
	BERTOverflow	81.47%	82.25%	81.86%
Template 2	RoBERTa	83.68%	84.14%	83.91%
	BERT	84.12%	83.58%	83.85%
	BERTOverflow	81.97%	81.24%	81.60%
Template 3	RoBERTa	84.29%	84.98%	84.63%
	BERT	84.63%	83.86%	84.24%
	BERTOverflow	83.11%	82.33%	82.72%
Template 4	RoBERTa	83.84%	83.19%	83.51%
	BERT	84.76%	83.47%	84.11%
	BERTOverflow	82.58%	82.56%	82.57%

Table 3. Performance of three models with and without LoRA

	RoBERTa	BERT	BERTOverflow
Without LoRA	12m 20s	11m 15s	10m 20s
With LoRA	7m 14s	6m 32s	5m 12s

5.2 Efficiency of API review classification by fine-tuning

In this RQ, we investigate the performance and efficiency of API review classification by using LoRA fine-tuning. Table 3 shows the average training time across the 9 categories for the three pre-trained models based on the template 3. From the table, we can find that LoRA significantly reduces the training time, indicating that it optimizes computational efficiency. For example, the training time with RoBERTa shows a 42% speed-up (from 12m 20s to 7m 14s), and the training time with BERTOverflow achieves a 50% improvement (from 10m 20s to 5m 12s). This suggests that LoRA fine-tuning reduces the training overhead due to its efficient parameter tuning approach that avoids full model fine-tuning. We also compare the performance of classification of RoBERTa model with and without fine-tuning in Table 4. We can find that with LoRA, the performance slightly decreases by 4% (e.g., from 84.63% to 80.69% in terms of F1 score). This trade-off suggests that while LoRA leads to a small reduction in predictive performance, it can improve training efficiency for the API review classification. Such finding can provide the guidance that LoRA can be preferable for applications where speed is more critical than minor accuracy losses.

Table 4. Efficiency of classification with and without LoRA for RoBERTa

	Avg Precision	**Avg Recall**	**Avg F1 Score**
Without LoRA	84.29%	84.98%	84.63%
With LoRA	80.13%	81.25%	80.69%

6 Threats to Validity

The main external threat to the validity is the dataset we used. In our study, we use the widely used dataset collected by Uddin et al. [13] for API review classification. But the dataset is imbalance, leading to the model misinterpreting the words from the beginning. In future, more dataset can be used to evaluate the performance of prompt-based API review classification.

7 Conclusion

In this paper, we conducted a comprehensive study to evaluate the performance of prompt-based API review classification by designing various prompt templates on pre-trained models. Our experimental results show that prompts with complete context can achieve best effectiveness and the model RoBERTa outperforms other two models due to the size of training corpus. Also, LoRA fine-tuning can achieve similar performance but significantly reduce the training overhead.

References

1. Liu, Yinhan and Ott, Myle and Goyal, Naman and Du, Jingfei and Joshi, Mandar and Chen, Danqi and Levy, Omer and Lewis, Mike and Zettlemoyer, Luke and Stoyanov, Veselin. Roberta: A robustly optimized bert pretraining approach, arXiv preprint arXiv:1907.11692, 2019
2. Uddin, Gias and Robillard, Martin P. How API documentation fails. Ieee software. 2015.
3. Uddin, Gias and Khomh, Foutse. Mining api aspects in api reviews. Technical report.2017
4. Uddin, Gias and Khomh, Foutse. Automatic summarization of API reviews. 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). 2017
5. Lin, Bin and Zampetti, Fiorella and Bavota, Gabriele and Di Penta, Massimiliano and Lanza, Michele. Pattern-based mining of opinions in q&a websites. 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). 2019
6. Yang, Chengran and Xu, Bowen and Khan, Junaed Younus and Uddin, Gias and Han, Donggyun and Yang, Zhou and Lo, David. Aspect-based api review classification: How far can pre-trained transformer model go? 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). 2022
7. Zhang, Yingying and Hou, Daqing. Extracting problematic API features from forum discussions. 2013 21st International Conference on Program Comprehension (ICPC). 2013
8. Ahasanuzzaman, Md and Asaduzzaman, Muhammad and Roy, Chanchal K and Schneider, Kevin A. Classifying stack overflow posts on API issues. 2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER). 2018
9. Treude, Christoph and Robillard, Martin P. Augmenting API documentation with insights from stack overflow. Proceedings of the 38th International Conference on Software Engineerin. 2016
10. Feng, Zhangyin and Guo, Daya and Tang, Duyu and Duan, Nan and Feng, Xiaocheng and Gong, Ming and Shou, Linjun and Qin, Bing and Liu, Ting and Jiang, Daxin and others. Codebert: A pre-trained model for programming and natural languages. 2020
11. Zeng, Zhengran and Tan, Hanzhuo and Zhang, Haotian and Li, Jing and Zhang, Yuqun and Zhang, Lingming. An extensive study on pre-trained models for program understanding and generation. Proceedings of the 31st ACM SIGSOFT international symposium on software testing and analysis. 2022
12. Xia, Chunqiu Steven and Wei, Yuxiang and Zhang, Lingming. Automated program repair in the era of large pre-trained language models. 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). 2023
13. Uddin, Gias and Khomh, Foutse. Automatic mining of opinions expressed about apis in stack overflow. IEEE Transactions on Software Engineering. 2019
14. Tabassum, Jeniya and Maddela, Mounica and Xu, Wei and Ritter, Alan. Code and named entity recognition in stackoverflow. arXiv preprint arXiv:2005.01634. 2020
15. Devlin, Jacob. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805
16. Qiu, Xipeng and Sun, Tianxiang and Xu, Yige and Shao, Yunfan and Dai, Ning and Huang, Xuanjing. Pre-trained models for natural language processing: A survey. Science China technological sciences, 2020
17. Liu, Yang and Zhang, Yao and Wang, Yixin and Hou, Feng and Yuan, Jin and Tian, Jiang and Zhang, Yang and Shi, Zhongchao and Fan, Jianping and He, Zhiqiang. A survey of visual transformers. IEEE Transactions on Neural Networks and Learning Systems, 2023
18. Brown, Tom B. Language models are few-shot learners. arXiv preprint arXiv:2005.14165. 2020
19. Schick, Timo and Schütze, Hinrich, Exploiting cloze questions for few shot text classification and natural language inference. arXiv preprint arXiv:2001.07676. 2020
20. Luo, Xianchang and Xue, Yinxing and Xing, Zhenchang and Sun, Jiamou. Prcbert: Prompt learning for requirement classification using bert-based pretrained language models. Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, 2022

21. Sahoo, Pranab and Singh, Ayush Kumar and Saha, Sriparna and Jain, Vinija and Mondal, Samrat and Chadha, Aman. A systematic survey of prompt engineering in large language models: Techniques and applications. arXiv preprint arXiv:2402.07927,2024
22. Rahimi, Nouf and Eassa, Fathy and Elrefaei, Lamiaa. One-and two-phase software requirement classification using ensemble deep learning. Entropy, 2021
23. Loshchilov, I. Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101. 2017