An Empirical Study of Adversarial Training in Code Comment Generation

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

Xiaolin Ju*
Nantong University
China
ju.xl@ntu.edu.cn

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

Guang Yang
Nanjing University of Aeronautics
and Astronautics, China
novelyg@outlook.com

***Abstract*—The code comment generation task is designed for developers to understand programs more quickly during development and maintenance. However, the existing automatic code comment generation models can not generate valuable comments for developers. It is necessary to explore a technology that can optimize the performance of code comment generation models without changing the model. We consider adversarial training as the experimental object, which can improve the robustness and generalization of the model. We present a large-scale study to experimentally validate the performance of gradient-based adversarial training methods in the code comment generation task. The results show that adversarial training can improve the model performance by generating adversarial examples without changing the model. Our empirical study can provide a new perspective for researchers to improve the performance of code comment generation models.**

***Index Terms*—Code comment generation, Adversarial Training, Bash code, Deep Learning, Empirical Study**

I. Introduction

Research shows that software developers and maintainers spend 59% [1] of their time on program understanding. A good code comment can improve the efficiency of software development and maintenance. Automating the generation of comments for code is a pressing issue. With the development of deep learning techniques and pre-trained models, the task of automatic code comment generation has also achieved SOTA (state-of-the-art) in academic research [2], [3]. However, recent empirical studies have shown that comments automatically generated for code based on existing models [3], [4] are ineffective in guiding developers. For example, Mastropaolo et al. [5] found that only 10% of the comments could reach the level of human writing. This suggests that software comment generation is still a development away from full industrial use.

For this reason, we consider using some techniques to optimize the model performance without changing the model, and adversarial training is a good way. Some studies used rule-based adversarial training methods in NLP (Natural Language Processing) tasks. For example, Ribeiro et al. [6] found that adversarial attacks can be useful when debugging NLP models. Zhang et al. [7] also found that model performance can be significantly improved in the field of code comment generation by generating adversarial examples. Different adversarial rules have been designed for various natural language processing (NLP) tasks. In this context, we focus on gradient-based adversarial training methods [8]–[11] and present a large-scale study evaluating the impact of adversarial training on code comment generation. Previous studies have not investigated whether gradient-based adversarial training methods can optimize the performance of code comment generation models.

In our empirical study, we explore the performance of gradient-based adversarial training methods on different code annotation generation models and different adversarial training methods on the same model. We considered four classic adversarial training methods: FGSM [8], FGM [9], PGD [10], and FreeLB [11]. Moreover, we also consider three types of code comment generation methods: deep learning-based [12], [13], pre-trained model-based [3], [4], [14], and hybrid model [5], [15], [16].

In summary, the contributions of our empirical study can be summarized as follows.

- To the best of our knowledge, we are the first to investigate improving the performance of code comment generation models by adversarial training.
- We conducted a large-scale empirical study to investigate this issue. In our empirical study, we selected five adversarial training methods and three types of code comment generation models. Our empirical results validate that adversarial training can improve the performance of code comment generation models. Moreover, we find that the PGD method is optimum for evaluating different adversarial training methods.
- To let other researchers follow our research, we shared our scripts, datasets, and results on the project homepage [1].

II. Related Work and Research Motivation

In this section, we first analyze the work related to the task of adversarial training and code comment generation. After analyzing the relevant research, we emphasize the novelty of this research.

A. Code Comment Generation

Code comment generation [2], [5], [12] can be defined as a code understanding and neural machine translation problem.

* Corresponding author.

DOI reference number: 10.18293/SEKE2023-108.

[1]https://github.com/syhstudy/AT_Empirical_Study

We classify these studies into information retrieval-based, deep learning-based, and hybrid methods. The method based on information retrieval is applicable to code with high reusability. Wong et al. [17] used SIM (a token-based code cloning detection tool) to detect stack overflow code fragments and corresponding descriptions, which used the detected pseudocode as the final comment. Recently, Yang et al. [18] proposed the method CCGIR, which retrieves smart contract codes based on semantic similarity, lexical similarity, and syntactic similarity. In terms of deep learning-based methods, Allamanis et al [19] put forward a model, which uses CNN and attention mechanism to detect the attention characteristics of local time-invariant and remote topics, and uses GRU to decode the output. Recently, Yang et al. [2] proposed a new Transformer-based method ComFormer and a fusion method based on mixed code representation. The hybrid method takes advantage of both information retrieval and deep learning. Li et al. [20] combined the code comments obtained by information retrieval with the semantic information of the input code to generate code comments. Recently, Yu et al. [16] proposed a hybrid method of two-stage training, which generates Bash comments through one-stage information retrieval and two-stage CodeBERT fine-tuning.

B. Adversarial Training in NLP

Adversarial training is an important way to enhance the robustness of neural networks. In the process of adversarial training, examples will be mixed with some small perturbations (the change is small, but it is likely to cause misclassification), and then the neural network will adapt to this change, thus being robust to adversarial examples.

The general principle of adversarial training can be summarized as the following maximum-minimum formula:

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

where x represents input, δ represents perturbation, y represents the label of the example, and $max(L)$ represents the optimization objective.

In the NLP (natural language processing) domain, typical adversarial training methods can be categorized as rule-based and gradient-based methods. In terms of rule-based methods, researchers generate adversarial examples by heuristic rules. Ebrahimi et al. [21] proposed a white-box method, which uses character/word substitution to generate adversarial examples. Ribeiro et al. [6] found that adversarial attacks can be useful when debugging NLP models. Cheng et al. [22] also found that model performance can be significantly improved in the field of neural machine translation by generating adversarial examples. In the type of gradient-based methods, researchers generate adversarial examples by adding a perturbation to the Embedding layer. Miyato et al. [9] introduced adversarial training and virtual adversarial training [23] to improve the performance of classification models. Zhu et al. [11] proposed FreeLB for the language model, which promotes higher invariance in the embedding space by adding hostile perturbation to the word embedding and minimizing the risk of result hostility in different regions around the input sample. Recently, Zhou et al. [24] achieved optimal performance using adversarial training and integrated learning techniques by two separately trained encoder-decoder models for source code sequences and corresponding abstract syntax trees (ASTs) in the area of code summarization generation.

C. Research Motivation

Research [5] shows that only 10% of the automatically generated comments can reach the human level. For this reason, we hope to explore a technology that can optimize the model performance without changing code comment generation models, and verify the feasibility of this exploration direction through preliminary results. We consider adversarial training as the experimental object, which can improve the robustness [25] and generalization [26] of the model, and can also be used as a technology for data augmentation [27]. However, to the best of our knowledge, we haven't found any work related to adversarial training in code comment generation task. To fill this gap, we have done novel experiments to explore the performance of adversarial training in code comment generation.

III. Research Questions

In our empirical study, we aim to answer the following three research questions (RQs).

RQ1: How efficient is adversarial training on classic deep learning models?

Motivation. The classical deep learning model is a highly recognized model in earlier studies, and it is often used as a baseline in current studies [16], [28]. We consider them as experimental objects, which can better reflect the rigor of our experiments.

RQ2: How efficient is adversarial training on pre-trained models?

Motivation. Code comment generation task is a typical text-to-text problem and PLMs (pre-trained language models) [3], [4], [14] have achieved SOTA in the current task, which is the focus of current researchers. We consider them as experimental objects, which can better reflect the universality of confrontation training.

RQ3: How efficient is adversarial training on hybrid models?

Motivation. The hybrid models [5], [15], [16] use deep learning and information retrieval technology to enhance the model representation through retrieval. We consider them as experimental objects, which can better reflect the comprehensiveness of our experimental setup.

IV. Case study design

In this section, we describe our experimental setup to address our research questions. We show the overview of our experiment design in Figure 1. This figure mainly shows the process of building code comment generation models through adversarial training method.

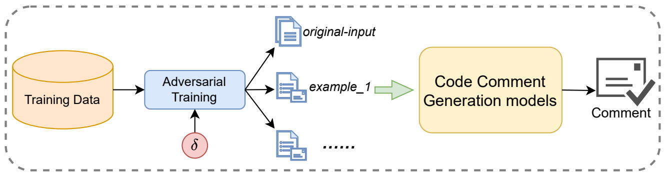

Fig. 1. Overall framework of our experiment design.

A. Experimental Subject

We consider the corpus shared by Yu et al. [16] as the experimental object in our research and continue to do research on more corpus in the future. This high-quality corpus contains 10,592 samples from NL2Bash [29] and NLC2CMD competition[2]. The statistics of corpus are shown in TABLE I.

TABLE I
LENGTH STATISTICS OF SAMPLES IN THE CORPUS

Code length statistics					
Average	Mode	Median	<16	<32	<48
8.528	4	7	90.8%	99.7%	99.9%
Code comment length statistics					
Average	Mode	Median	<16	<32	<48
11.874	10	11	80.3%	99.5%	99.9%

In this table, we find that Bash codes and comments are mostly within 48. In the corpus division, we use the random sampling method [28] to randomly divide the corpus into the training set, evaluation set, and test set according to the ratio of 8:1:1.

B. Performance Measures

In order to quantitatively compare the performance of each code comment generation method, we consider three performance metrics (BLEU [30], METEOR [31] and ROUGE-L [32]) from the research of neural machine translation. These performance metrics are also widely used in the research of automatic code comment generation [33], which can compare the quality of generated comments and reference comments. The higher the calculated performance metric, the better the performance of the corresponding method.

We use the implementation provided by the nlg-eval library[3] to evaluate the performance, which can avoid the difference in results caused by different experiments.

C. Adversarial Training Methods

Adversarial training generates adversarial examples x_{adv} by adding perturbation δ to model input x, where $x_{adv} = x + \delta$. In this experiment, we mainly consider the following four methods of adversarial training that are often used in NLP:

- **FGSM.** FGSM is a method of adversarial training proposed by Goodfellow et al. [8]. The input gradient is $g = \nabla_x L(f_\theta(x), y)$, where θ is the model parameter value, x is the model input, y is the label, and $L()$ is 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 is also the method proposed by Goodfellow et al. [9]. Unlike FGSM, which takes the $sign$ function to regularize the gradient, FGM carries out $L2$ regularization on the gradient. In addition, FSGM 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).
- **PGD.** FGM calculates the perturbation directly through epsilon parameter, which may not be optimal. Therefore, PGD [10] has been 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.
- **FreeLB.** In order to find the optimal perturbation, Zhu et al. [11] proposed the FreeLB method. PGD takes the gradient g_t of the last perturbation after iterating K times when updating parameters, and FreeLB takes the average gradient g_{avg} of K times of iteration when updating parameters, where $g_{avg} = (g_1 + \cdots + g_t)/t$.

D. Code Comment Generation Methods

To show the feasibility of our proposed research, we have recently considered three types of methods that researchers have focused on (i.e., classical deep learning method, pre-training model method, and hybrid method).

The first type is deep learning-based methods.

- **CODE-NN.** CODE-NN [12] is the first to generate code comments using LSTM and attention mechanism.
- **Transformer.** Transformer [13] is an encoder-decoder model that utilizes the self-attention mechanism.

The second type is pre-trained-based methods.

- **CodeBERT.** CodeBERT [34] builds the model based on the neural architecture of Transformer, and uses the mixed objective function to train the model.
- **UniXcoder.** UniXcoder [14] 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.
- **CodeT5.** CodeT5 [3] supports multi-task learning and can make better use of the information of code tokens to train the model.

The third type is the methods of hybrid information retrieval and deep learning techniques.

[2]https://eval.ai/web/challenges/challenge-page/674/leaderboard/1831
[3]https://github.com/Maluuba/nlg-eval

- **Rencos.** Rencos [15] first retrieves similar codes, and then the encoder vectors are fused by the decoder.
- **Hybrid-DeepCom.** Hybrid-DeepCom [5] considers the semantics of Java and traverses AST to obtain syntax information and structure information of the code.
- **BASHEXPLAINER.** BASHEXPLAINER [16] utilizes two-stage training strategies: fine-tuning stage and the information retrieval enhancement stage.

E. Experimental Settings

We implement our experiment based on Pytorch 2.0. Specifically, we choose AdamW as the optimizer and utilize HuggingFace [4] to implement our selective method. We set the value of epoch to 30 and set early stopping with epoch 5.

We run all the experiments on a computer with an Intel CPU i5-13600K and a GeForce RTX 4090 GPU with 24 GB memory. The running OS platform is Windows OS.

V. Result analysis

A. RQ1: How efficient is adversarial training on classic deep learning models?

TABLE II
Comparison between baselines of deep learning models whether to add PGD

Model Name	BLEU-3	BLEU-4	METEOR	ROUGE-L
CODE-NN	29.53 **32.15**	24.17 **27.55**	26.85 **27.98**	47.21 **49.44**
Transformer	25.42 **30.22**	19.97 **27.03**	25.22 **26.37**	44.01 **46.18**

Table II shows the overall results of the different deep learning methods concerning three evaluation measures (i.e. BLEU, ROUGE-L, and METEOR), and we mark the best one of each metric in bold. The first line of the table is without PGD method, and the second line is with PGD method.

According to the experimental results, we find that adversarial training PGD can improve the performance of classic deep learning models. Specifically, compared with not using PGD, the performance of classic deep learning models is improved by 18.89%, 35.35%, 4.56%, and 4.93% for BLEU-3, BLEU-4, METEOR, and ROUGE-L at least. This result indicates that adversarial training, as a special regularization method [35], plays an important role in optimizing the model performance method.

Answer to RQ1: Comparison results show that adversarial training can improve the performance of classic deep learning models.

B. RQ2: How efficient is adversarial training on pre-trained models?

Table III shows the overall results of the different pre-trained methods concerning three evaluation measures (i.e. BLEU, ROUGE-L, and METEOR), and we mark the best one of each metric in bold. The first line of the table is without PGD method, and the second line is with PGD method.

[4]https://huggingface.co

TABLE III
Comparison between baselines of pre-trained models whether to add PGD

Model Name	BLEU-3	BLEU-4	METEOR	ROUGE-L
CodeBERT	29.84 **33.83**	24.83 **29.28**	27.16 **28.95**	47.36 **50.18**
UniXcoder	31.80 **33.64**	27.25 **29.12**	29.03 **29.68**	48.24 **49.26**
CodeT5	33.25 **34.70**	28.70 **29.93**	29.49 **30.29**	48.36 **50.56**

According to the experimental results, we find that adversarial training PGD can improve the performance of pre-trained models. Specifically, compared with not using PGD, the performance of pre-trained models is improved by 13.37%, 17.92%, 6.59%, and 5.95% for BLEU-3, BLEU-4, METEOR, and ROUGE-L at least. This result indicates that adversarial training, as a data augmentation method [27], greatly improves the pre-trained model that needs to be trained with a large-scale corpus.

Answer to RQ2: Comparison results show that adversarial training can improve the performance of pre-trained models.

C. RQ3: How efficient is adversarial training on hybrid models?

TABLE IV
Comparison between baselines of hybrid models whether to add PGD

Model Name	BLEU-3	BLEU-4	METEOR	ROUGE-L
Rencos	28.66 **30.22**	24.39 **27.70**	25.82 **26.83**	45.06 **47.48**
Hybrid-DeepCom	27.91 **30.47**	22.75 **26.09**	26.27 **27.33**	45.36 **47.55**
BASHEXPLAINER	33.73 **35.06**	29.13 **30.39**	28.78 **29.35**	48.81 **49.78**

Table IV shows the overall results of the different hybrid methods concerning three evaluation measures (i.e. BLEU, ROUGE-L, and METEOR), and we mark the best one of each metric in bold. The first line of the table is without PGD method, and the second line is with PGD method.

According to the experimental results, we find that adversarial training PGD can improve the performance of hybrid models. Specifically, compared with not using PGD, the performance of hybrid models is improved by 9.17%, 14.68%, 4.04%, and 5.37% for BLEU-3, BLEU-4, METEOR, and ROUGE-L at least. The result indicates that the model robustness improved by adversarial training [25] may be helpful to the model that needs multi-stage training.

Answer to RQ3: Comparison results show that adversarial training can improve the performance of hybrid models.

VI. Discussion & implications

In this section, we first study the effect of the adversarial training method considered on code comment generation models. Secondly, we introduce the implication of the experiment and propose a novel adversarial training method based on regularization optimization, and verify the feasibility of this exploration direction through preliminary results.

A. Analysis on Adversarial Training Methods

In this subsection, we want to explore which adversarial training method is suitable for code comment generation. We consider evaluating the performance of different adversarial training methods on CodeBERT. Table V shows the comparison results, and we mark the best one of each metric in bold. The first line of the table is without the adversarial training method, and the other lines are with the adversarial training method.

TABLE V
Comparison between different adversarial training (AT) methods on CodeBERT

AT Name	BLEU-3	BLEU-4	METEOR	ROUGE-L
FGSM	29.69	25.22	27.12	47.99
FGM	33.52	29.05	28.10	49.28
PDG	**34.55**	**29.98**	**29.31**	**49.86**
FreeLB	33.83	29.28	28.95	49.67

According to the experimental results, we find that the performance of CodeBERT can be improved to the maximum by using PGD method. Specifically, compared with the method with the lowest index, CodeBERT can improve the performance by 16.36%, 18.87%, 8.08%, and 3.90% for BLEU-3, BLEU-4, METEOR, and ROUGE-L respectively. This result indicates that only the last iteration of PGD method is more suitable for the code comment generation task.

B. Practical Guidelines

Our experimental results show that adversarial training technology can affect the performance of the model, especially for the complex pre-trained model. In this subsection, we provide practical guidelines for future research on code comment generation.

Researchers can consider improving the performance of code comment generation models from outside the model. The results in Section 3 show that adversarial training technology can significantly improve the performance of code comment generation models. As a special regularization method [35], adversarial training can optimize model parameters by generating adversarial examples without changing the model. We suggest that future researchers can optimize code comment generation models by using adversarial training technology.

Researchers can change the regularization method to make adversarial training more suitable for the code comment generation task. As mentioned in the previous paragraph, as a special regularization method [35], the optimization loss function can be expressed as $\widetilde{L}(x, y) \approx L(x, y) + \frac{\epsilon}{2}\left\|\partial_x L\right\|_q$, where x represents the input, y represents the label, $\widetilde{L}$ represents the loss after joining the adversarial training, L represents the original loss, and $\frac{\epsilon}{2}\left\|\partial_x L\right\|_q$ represents the special regularization term. This shows that the model performance can be further optimized by changing the regularization method. On this basis, we propose a novel adversarial training method, normPGD, which is oriented to the code comment generation task. Limited by space, the related introduction and results are on the project homepage [5]. We suggest that future researchers can customize the adversarial training method which is more suitable for the code comment generation task by changing the regularization term.

VII. Threats to Validity

Internal threats. The main first internal threat is the implementation correctness of code comment generation models we chose. To alleviate this threat, we re-implement their method according to the description of their empirical research and achieve similar performance. The second internal threat is the potential defect of the experimental model we designed. To alleviate this threat, we use mature libraries to implement code, such as Pytorch and Transformers.

External threats. The threat of external validity is the choice of experimental subjects. In order to alleviate this threat, we chose the corpus provided by Yu et al. [16]. Yu et al. improved the quality of data pairs by pre-processing the corpus. At the same time, this corpus has also been used in the research of code comment generation. In the future, we hope to verify the performance of our questions in popular languages (such as Java, Python, etc.).

Construct threats. The construct threat in this study is the performance measures used to evaluate our proposed experiments performance. In order to alleviate this threat, we choose three popular performance metrics (i.e. BLEU [30], METEOR [31], and ROUGE-L [32]) in the field of neural machine translation. These evaluation metrics are also widely used in the field of code comment generation [33].

Conclusion threats. The conclusion threat in our study is mainly that there is no cross-validation. Due to the high training cost of deep learning, this method has not been widely used in the field of neural machine translation. In this study, we only divide the corpus once, which is consistent with some previous studies on code comment generation [15].

VIII. Conclusion

This paper gives a comprehensive evaluation of the adversarial training methods in the field of code comment generation. Specifically, we first collect the existing popular adversarial training methods and find out the most suitable method for the code comment generation task through experiments. Then, we propose three types of research questions and conduct experiments on them. Finally, we draw a conclusion about the performance of adversarial training in the code comment generation task. We believe that this study can help to further evaluate and improve the code comment generation task. The

[5]https://github.com/syhstudy/AT_Empirical_Study

limitation of this work lies in the limited corpus, models, and metrics. This can draw the help of adversarial training as a data augmentation method better for low-resource tasks. Future work may be carried out by collecting more corpus or evaluating more popular (such as Java, Python, etc.) corpus to more accurately evaluate the performance of adversarial training in the code comment generation task.

Acknowledgements

This work is supported in part by the Jiangsu Province Modern Educational Technology Research Project under Grant No. 2022-R-98984 and the Nantong Application Research Plan under Grant No. JCZ21087.

References

[1] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, "Measuring Program Comprehension: A Large-Scale Field Study with Professionals," *IEEE Trans. Software Eng.*, vol. 44, no. 10, pp. 951–976, Jul. 2017.

[2] G. Yang, X. Chen, J. Cao, S. Xu, Z. Cui, C. Yu, and K. Liu, "Comformer: Code comment generation via transformer and fusion method-based hybrid code representation," in *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*. IEEE, 2021, pp. 30–41.

[3] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 8696–8708.

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

[5] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation with hybrid lexical and syntactical information," *Empirical Software Engineering*, vol. 25, pp. 2179–2217, 2020.

[6] M. T. Ribeiro, S. Singh, and C. Guestrin, "Semantically equivalent adversarial rules for debugging nlp models," in *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

[7] X. Zhang, Y. Zhou, T. Han, and T. Chen, "Training deep code comment generation models via data augmentation," in *12th Asia-Pacific Symposium on Internetware*, 2020, pp. 185–188.

[8] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *3th International Conference on Learning Representations, ICLR 2015*, 2015.

[9] T. Miyato, A. M. Dai, and I. Goodfellow, "Adversarial training methods for semi-supervised text classification," *5th International Conference on Learning Representations, ICLR 2017*, 2017.

[10] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[11] C. Zhu, Y. Cheng, Z. Gan, S. Sun, T. Goldstein, and J. Liu, "Freelb: Enhanced adversarial training for natural language understanding," *8th International Conference on Learning Representations, ICLR 2020*, 2020.

[12] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Summarizing Source Code using a Neural Attention Model," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics: Long Papers*. Association for Computational Linguistics, Aug. 2016, pp. 2073–2083.

[13] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "A transformer-based approach for source code summarization," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 4998–5007.

[14] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "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)*, 2022, pp. 7212–7225.

[15] J. Zhang, X. Wang, H. Zhang, H. Sun, and X. Liu, "Retrieval-based neural source code summarization," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 1385–1397.

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

[17] E. Wong, J. Yang, and L. Tan, "Autocomment: Mining question and answer sites for automatic comment generation," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2013, pp. 562–567.

[18] G. Yang, K. Liu, X. Chen, Y. Zhou, C. Yu, and H. Lin, "Ccgir: Information retrieval-based code comment generation method for smart contracts," *Knowledge-Based Systems*, vol. 237, p. 107858, 2022.

[19] M. Allamanis, H. Peng, and C. Sutton, "A convolutional attention network for extreme summarization of source code," in *International conference on machine learning*. PMLR, 2016, pp. 2091–2100.

[20] J. Li, Y. Li, G. Li, X. Hu, X. Xia, and Z. Jin, "Editsum: A retrieve-and-edit framework for source code summarization," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 155–166.

[21] J. Ebrahimi, A. Rao, D. Lowd, and D. Dou, "Hotflip: White-box adversarial examples for text classification," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2018, pp. 31–36.

[22] Y. Cheng, L. Jiang, and W. Macherey, "Robust neural machine translation with doubly adversarial inputs," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 4324–4333.

[23] T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii, "Virtual adversarial training: a regularization method for supervised and semi-supervised learning," *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 8, pp. 1979–1993, 2018.

[24] Z. Zhou, H. Yu, and G. Fan, "Adversarial training and ensemble learning for automatic code summarization," *Neural Computing and Applications*, vol. 33, no. 19, pp. 12 571–12 589, 2021.

[25] P. Bielik and M. Vechev, "Adversarial robustness for code," in *International Conference on Machine Learning*. PMLR, 2020, pp. 896–907.

[26] J. Y. Yoo and Y. Qi, "Towards improving adversarial training of nlp models," in *Findings of the Association for Computational Linguistics: EMNLP 2021*, 2021, pp. 945–956.

[27] J. Morris, E. Lifland, J. Y. Yoo, J. Grigsby, D. Jin, and Y. Qi, "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*, 2020, pp. 119–126.

[28] G. Yang, Y. Zhou, X. Chen, and C. Yu, "Fine-grained pseudo-code generation method via code feature extraction and transformer," in *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2021, pp. 213–222.

[29] X. V. Lin, C. Wang, L. Zettlemoyer, and M. D. Ernst, "Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[30] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[31] S. Banerjee and A. Lavie, "Meteor: An automatic metric for mt evaluation with improved correlation with human judgments," in *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005, pp. 65–72.

[32] L. C. ROUGE, "A package for automatic evaluation of summaries," in *Proceedings of Workshop on Text Summarization of ACL, Spain*, 2004.

[33] B. Wei, Y. Li, G. Li, X. Xia, and Z. Jin, "Retrieve and refine: exemplar-based neural comment generation," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 349–360.

[34] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," *ACL Anthology*, pp. 1536–1547, Nov. 2020.

[35] C.-J. Simon-Gabriel, Y. Ollivier, L. Bottou, B. Schölkopf, and D. Lopez-Paz, "First-order adversarial vulnerability of neural networks and input dimension," in *International conference on machine learning*. PMLR, 2019, pp. 5809–5817.