CCGRA: Smart Contract Code Comment Generation with Retrieval-enhanced Approach

Zhenhua Zhang
College of Intelligence and Computing
Tianjin University
Tianjin, China
zzhbible@tju.edu.cn

Shizhan Chen
College of Intelligence and Computing
Tianjin University
Tianjin, China
shizhan@tju.edu.cn

Guodong Fan
College of Intelligence and Computing
Tianjin University
Tianjin, China
guodongfan@tju.edu.cn

Guang Yang
The College of Computer Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing, China
novelyg@outlook.com

Zhiyong Feng*
College of Intelligence and Computing
Tianjin University
Tianjin, China
zyfeng@tju.edu.cn

***Abstract*—Smart contracts are self-executing programs on the blockchain that are critical to a range of industries, including finance, supply chain management, and healthcare. However, comprehending smart contracts can be challenging due to a lack of effective comments in most user-defined code. To address this challenge, we propose a novel retrieval-enhanced approach CCGRA that leverages retrieval knowledge to generate high-quality comments for Solidity language code. Our approach carefully eliminates duplicated data and template data in the widely-used smart contract dataset to ensure a high-quality corpus. Extensive experiments and comprehensive analysis demonstrate the effectiveness applicability of our approach after being compared with eight state-of-the-art baselines. Finally, we conduct a human study and find the comment quality generated by our approach is better than baselines in terms of similarity, naturalness, and informativeness.**

***Index Terms*—Code Comment Generation, Smart Contract, CodeT5, Information Retrieval**

I. Introduction

Smart contracts are self-executing programs that reside on the blockchain. They comprise of template code and user-defined code. Template code provide basic functionalities such as asset transfer, voting, or escrow, while user-defined code tailors the smart contract to specific needs, including business logic and rules. However, user-defined code is often not commented, making it challenging for traders to read, especially in high-stakes transactions [1].

Adding precise comments to user-defined code is critical for contract code comprehensibility and trust-building. However, the quality of the dataset used for training comment generation models can significantly impact their ability to accurately capture the nuances of user-defined code [2]. Our investigation highlights that the Solidity datasets [3], which widely used for smart contract code comment generation tasks, contains a significant amount of template code data that can skew the model's learning towards the template code. This, in turn, compromises the model's ability to generate high-quality comments for user-defined code. To address this challenge, we propose a solution that involves removing duplicate template code comments, which can lead to more effective and accurate comment generation for user-defined code.

In this paper, we propose a novel retrival-enhanced approach CCGRA (Code Comment Generation with Retrival-enhanced Approach). Our approach leverages the advantages of pre-trained language models and retrieval techniques to generate reliable and high-quality comments for smart contract. To achieve this objective, we utilize the widely-used smart contract dataset [3], carefully eliminating duplicated data and template data to ensure a high-quality training set. In the end, the dataset we used contains 29,720 ⟨code, comment⟩pairs, and the experimental results indicate that it outperforms eight state-of-the-art baselines. Our approach showcases the potential of combining pre-trained language models and retrieval techniques to improve the quality of generated comment in the smart contract.

The main contributions can be summarized as follows:

- We address the challenges associated with current dataset used for smart contract code comment generation by eliminating duplicated data and template data.
- We conduct a comprehensive empirical study and a human evaluation to evaluate the performance of various baselines and demonstrate that our proposed approach, CCGRA, achieves state-of-the-art results.
- We share our corpus and scripts on our project homepage [1] to promote the replication of our research.

DOI reference number: 10.18293/SEKE2023-090
*Corresponding Author

[1]https://github.com/ZZHbible/CCGRA

II. Related Work

A. Code Comment Generation

1) Information retrieval-based approaches: In the early study phase, the researchers aimed to retrieve similar code from the software repository to improve the quality of code comments. Haiduc et al. [4], [5] considered two information retrieval models VSM and LSI. Rodeghero et al. [6] utilized the eye-tracking technique to identify the statements and keywords focused on by the developers.

2) Deep learning-based approaches: Recently, most of the previous studies followed deep learning-based approaches and achieved promising results. For example, Hu et al. [7] proposed the approach DeepCom by analyzing abstract syntax trees (ASTs). Later, Hu et al. [8] further proposed the improved approach Hybrid-DeepCom. Ye et al. [9] and Yang et al. [10] exploited the probabilistic correlation between the code summarization and code generation task via dual learning.

Ahmad et al. [11] used the Transformer model to generate code comments. The Transformer model is a kind of Seq2Seq model based on multi-head self-attention, which could effectively capture long-range dependencies. Then the first pre-trained model CodeBert on the source code was proposed by Feng et al. [12]. Later, Ahmad et al. [13] proposed another pre-training model PLBART for the source code. Recently, Wang et al. [14] proposed the pre-training model CodeT5.

3) Retrieval-enhanced approaches: Except for the approaches based on information retrieval or deep learning, recent studies also proposed approaches, which could fuse information retrieval and deep learning. Zhang et al. [15] were the first to take advantage of both information retrieval and deep learning approaches and proposed the approach Rencos. Wei et al. [16] used the comment of the similar code snippet as exemplars and proposed the approach Re^{2}Com.

B. Code Intelligence in Smart Contract

1) Vulnerability detection and repair: Code intelligence in smart contracts is its ability to detect and prevent security vulnerabilities. Code intelligence can analyze the code and identify potential security flaws, such as re-entrancy attacks, and recommend appropriate measures to mitigate them. Tsankov et al. [17] release static analysis tools named Securify used to analyze code vulnerabilities in smart contracts. Zhang et al. [18] proposed a system for automatically repairing code.

2) Smart contract code comment generation: The comment generation of smart contracts can effectively help traders understand the contract content. Shi et al. [19] propose an automated translation approach based on AST, and leveraged reinforcement learning to train a syntax synthesizer to generate comprehensible comments. Hu et al. [20] exploited the Transformer and Pointer mechanism to learn the representation of source code and generates natural language descriptions.

In this study, our main focus is on combining the retrieval augmentation method with a pre-training model, leveraging the existing code library and the pre-training model generalization ability for generating comments for smart contract code.

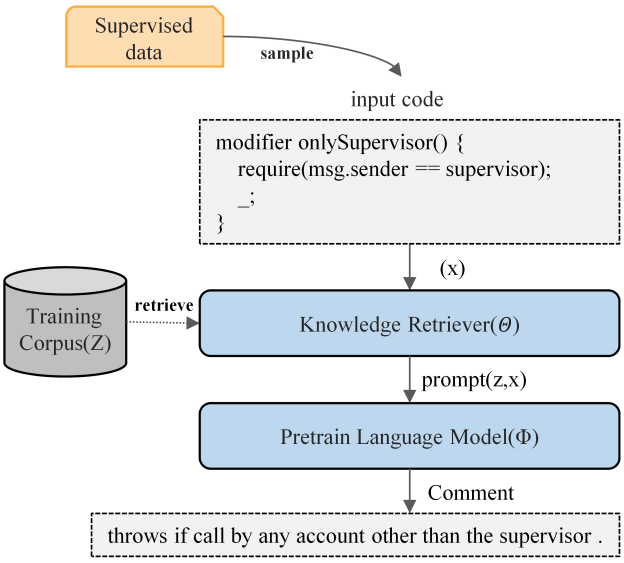


Fig. 1. The framework of CCGRA

III. Approach

In this section, we introduce the framework of CCGRA, which is illustrated in Fig. 1. Overall, CCGRA consists of two modules: (a) **Retrieval Module.** This module retrieves the most similar candidate code from the corpus to the given code. The comment associated with the retrieved code is then used as the basis for generating comment on the given code. (b) **Generation Module.** This module combines the retrieved comment with the given code, adds a prompt, and learns to generate the comment using the prior knowledge of a pre-trained language model.

A. Retrieval Module

Suppose we index the corpus into a list of key-value pairs, i.e. $\mathcal{Z} = \{(x_i, y_i)\}$, where x means the code and y means the comment. Then, given the input code x, the retrieval module Θ matches it with all code and returns the most similar code together with its comment:

$$\Theta(x \mid \mathcal{Z}) = \{(x_i', y_i')\} \tag{1}$$

In this work, we build the retrieval engine based on the CCGIR [21]. Therefore, for the given input code x, we define the input code sequence $\{x_i\}_{i=1}^{M}$, where M is the length of the code sequence. Then we encode the sequence via CodeBert, extract the hidden states to get the semantic vector $X \in \mathbb{R}^D$, which D denotes the hidden dimension of CodeBert. Then we further perform a linear transformation of X via BERT-whitening get $\tilde{X} \in \mathbb{R}^d$, which can reduce its dimension from D to d. Thus for the target code snippet x and the code snippet x_i in the corpus, we can get the semantic vectors $\tilde{X}$ and $\tilde{X}_i$ respectively. Then we can calculate their dot product score as their semantic similarity to select the $top-k$ most similar code snippets as the candidates from the corpus $\mathcal{Z}$.

To better combine syntactic and lexical knowledge of the source code, we separately utilize syntactic-level similarity and lexical-level similarity to find the most similar code x'.

Since the computational cost of calculating the similarities based on tree matching algorithms is high, we generate its corresponding AST sequence for each code snippet and then calculate the syntactical-level similarity via the edit distance. Since keywords occur more frequently than other tokens in the source code, these duplicated keywords may have a negative impact. We calculate the lexical-level similarity based on the set structure and Jaccard score.

Finally, we use $mixed\_score$ to retrieve the most similar code. For the code snippets x_1 and x_2, $mixed\_score$ can be calculated as follows.

$$\begin{aligned} mixed\_score\left(x_1, x_2\right) = \lambda \times lexical\_similarity \\ \left(x_1, x_2\right) + (1-\lambda) \times \text{syntactic\_similarity}\left(x_1, x_2\right) \end{aligned} \tag{2}$$

where λ is a hyper-parameter for knowledge fusion, which can control the ratio of the lexical-level similarity and the syntactical-level similarity.

B. Generation Module

For the given input code x, we use the retrieval engine to find the the most similar code x' together with its comment y'. As retrieval from a large corpus is computationally costly, we propose to retrieve from the labeled training data. In other words, we directly adopt the training data $\mathcal{T} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ as the indexed corpus $\mathcal{Z}$, where x_i is the input code and y_i is the ground-truth comment. Note that during training, as the input code x is already indexed, we filter it from the retrieval results to avoid data leakage.

Inspired by Instruct-GPT [22], constructing instructions for downstream tasks can stimulate the potential of pre-trained models. For our task, we design the template function f by appending task-specific instructions as follows.

$$f(x, y') = \text{“}summarize\ Solidity\ : y' \oplus x\text{”} \tag{3}$$

Here we use $f(x, y')$ to denote the input of the pre-trained model, and we use CodeT5 [14] as the backbone model for our task. The encoder inputs $f(x, y')$ and outputs the hidden representation $h = Enc(f(x, y'))$. Then the decoder iterates on the previously generated token $y < j$ via self-attention, and then predicts the probability of the next text token $P_\Phi(y_j|y_{<j}, x) = Dec(y_{<j}, h)$. We train our model Φ by minimizing the negative log-likelihood of the target text tokens y for a given input $f(x, y')$. The formula can be defined as follows

$$\mathcal{L}_\Phi = -\sum_{j=1}^{|y|} \log P_\Phi\left(y_j \mid y_{<j}, f(x, y')\right) \tag{4}$$

IV. Experimental Setup

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

RQ1: How effective is our CCGRA compared to the baseline models in terms of automatic performance measures?

RQ2: How do various retrieval methods affect the retrieval-augmented pre-training model?

RQ3: How effective is our CCGRA at generating higher-quality comments in terms of human evaluation?

A. Experimental Subjects

In our empirical study, we employed a carefully selected corpus of smart contracts sourced from Etherscan.io and provided by Zhuang et al. [3] and Yang et al. [21]. To ensure a high-quality dataset, we carefully eliminated duplicate data and template data. Our final dataset consisted of 29,720 pairs of data, which we split into training, validation, and testing sets in an 8:1:1 ratio. We also computed the average number of tokens in both code and comments, providing detailed statistics in Table I.

TABLE I
Statics Of The Datasets

Type	Train	Validation	Test
Count	23,776	2,972	2,972
Avg. tokens in code	80.54	80.13	82.27
Avg. tokens in comment	12.05	11.97	12.10

B. Performance Evaluation Measures

In our experimental study, we use three performance evaluation measures (i.e., $BLEU$ [23], $METEOR$ [24], and $ROUGE\text{-}L$ [25]) from the source code summarization domain to automatically evaluate the quality of the generated comments. Moreover, these performance measures have also been widely used in previous studies for source code summarization [26], which can alleviate the construct threats of our empirical study.

To avoid the result difference due to different performance measure implementation versions [27], we utilize the nlg-eval package[2], which can ensure the implementation correctness of these performance measures and guarantee a fair comparison.

C. Baselines

To show the competitiveness of our proposed approach CCGRA, we evaluate our proposed approach against eight state-of-the-art source code summarization baselines. Specifically, we classify these baselines into three groups. The first group is information retrieval approaches, including BM25 [28], NNGen [29], and CCGIR [21]. The second group is deep learning approaches, including CodeBert [12], UniXcoder [13], and CodeT5 [14]. The last group is hybrid approaches, including Rencos [15] and BashExplainer [30].

D. Experimental Settings

In our empirical study, we use the packages Faiss[3] and Transformers[4] to implement our proposed approach CCGRA. The hyper-parameters and their values in our empirical study are summarized in Table II.

All the experiments run on a computer with an Intel(R) Xeon(R) Silver 4210 CPU and a GeForce RTX3090 GPU with 24 GB memory. The running OS platform is Ubuntu operation system.

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

[3]https://github.com/facebookresearch/faiss

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

TABLE II
HYPER-PARAMETERS AND THEIR VALUES IN OUR EMPIRICAL STUDY

Module	Hyper-parameter	Value
Retrieval	Dimension d after BERT-whitening	256
	Number k for top-k candidates	5
	Coefficient α of $mixed\_score$	0.7
Generation	Model size d_{model}	768
	Number of layers N	12
	Number of multi-attention-heads	12

V. EXPERIMENTAL RESULTS

A. RQ1: How effective is our CCGRA compared to the baseline models in terms of automatic performance measures?

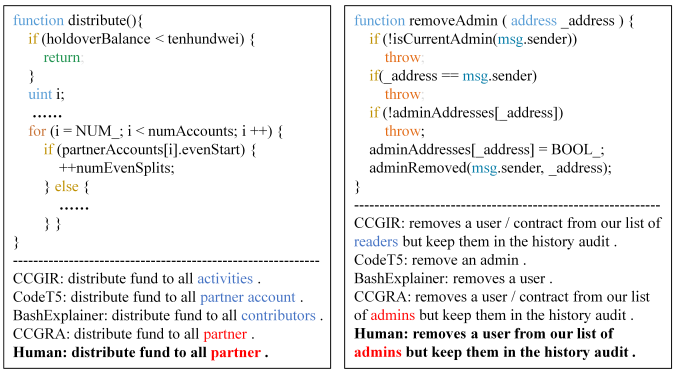
In this RQ, we want to investigate how effective our approach is and how much performance improvement our approach can achieve over the baselines.

TABLE III
COMPARISON BETWEEN BASELINES AND CCGRA

Model Name	BLEU-3	BLEU-4	METEOR	ROUGE-L
BM25	20.18	16.98	16.81	37.08
NNGen	21.33	18.17	16.83	37.70
CCGIR	22.12	18.96	17.33	38.12
CodeBert	19.34	16.61	17.38	42.15
UniXcoder	19.49	16.56	17.03	40.46
CodeT5	23.16	20.48	19.82	45.86
Rencos	17.35	14.53	14.95	38.45
BashExplainer	21.09	18.62	18.22	42.62
CCGRA	**25.55**	**22.20**	**20.84**	**46.32**

The results are shown in Table III where the best results are in bold fonts. We can find that we proposed CCGRA achieve the best results among the eight baselines. By comparing the average growth rate of four metrics with the baseline models, our proposed CCGRA outperforms them by a substantial margin of 26.58%, 22.17%, 18.58%, 23.88%, 25.49%, 6.21%, 30.12%, and 9.57% for BM25, NNGen, CCGIR, CodeBert, UniXcoder, Rencos, and BashExplainer, respectively. And the results indicate that retrieval-based methods outperform the CodeBert and UniXcoder models in terms of BLEU scores, suggesting that the comments generated by retrieval-based methods are effective in maintaining language consistency. However, in terms of ROUGE-L scores, pre-trained models perform better than retrieval-based methods, indicating that the comments generated by pre-trained models match the reference comments more closely in terms of semantic similarity. Moreover, it can be observed that the transformer with an encoder-decoder structure has better performance in generating Solidity comment (which does not appear in the pre-training data) compared to models with a single encoder structure.

To further demonstrate the effectiveness of CCGRA, we conduct qualitative analysis and two examples of generated comments are listed in Fig. 2. We denote comments written by humans in bold black font, and highlight comparisons



(a) Example 1 (b) Example 2

Fig. 2. Examples of generated comments by CCGRA and other baselines

in light blue and red. From Fig. 2(a), we find that through comparing the light blue and red comments that CCGIR and BashExplainer generated "activities" and "contributors" respectively, which is not correct. Meanwhile, CodeT5 generated "partner account" which is not as accurate as human-written comments. Only CCGRA is more accurate in identifying the objects in generating comments compared to the approaches of retrieval and pre-trained models. From Fig. 2(b), we find that when generating long text comments, the comments generated by retrieval-based approaches are not accurate enough, and those generated by pre-trained models are relatively short and lacked comprehensiveness. However, CCGRA can effectively overcome the shortcomings of these approaches and generate comments that are most in line with human-written comments. CCGRA combines the advantages of retrieval-based approaches by retrieving similar comment and constructing prompts to reduce semantic bias in generating comments on unseen datasets during pretraining, achieving optimal results.

> **Summary for RQ1:** CCGRA can achieve better performance than eight state-of-the-art baselines in automatic evaluation.

B. RQ2: How do various retrieval methods affect the retrieval-augmented pre-training model?

To show the relative importance of retrieval-based approach in CCGRA, we perform a series of ablation studies over the key modules.

TABLE IV
COMPARISON BETWEEN DIFFERENT BASELINES

Model Name	BLEU-3	BLEU-4	METEOR	ROUGE-L
UniXcoder	19.49	16.56	17.03	40.46
with BM25	21.36	18.23	18.82	44.32
with NNGen	21.56	18.56	18.34	43.89
with CCGIR	**23.16**	**20.48**	**19.82**	**45.86**
CodeT5	23.16	20.48	19.82	45.86
with BM25	24.42	21.01	20.52	46.02
with NNGen	23.86	20.72	20.05	45.56
with CCGIR	**25.55**	**22.20**	**20.84**	**46.32**

Table IV shows the performance comparison of several retrieval-based approaches combined with pre-trained models. We observe that an effective retrieval approach can promote the model to learn the representation of the code, thereby guiding the generation of appropriate comments by the model. Meanwhile, the results show that BM25, NNGen, and CCGIR can improve UniXcoder and CodeT5 by 10.08%, 12.08%, 23.67%, and 2.57%, 1.17%, 8.40%, respectively, in terms of the BLEU-4 score. Furthermore, our results show that an improved retrieval ability is positively correlated with the model's learning ability. We find that the performance of the retrieval module directly impacted the model's overall performance, which can highlight the positive impact of retrieval augment.

Summary for RQ2: The incorporation of a retrieval module into the overall model has a significant impact on its performance. Specifically, the performance of the retrieval module is positively correlated with the overall model's effectiveness.

C. RQ3: How effective is our CCGRA at generating higher-quality comments in terms of human evaluation?

Although automatic performance metrics can evaluate the gap between the generated comments and reference comments written by humans, these performance measures may not truly reflect the semantic similarity between different comments [31]. To verify the effectiveness of our proposed approach CCGRA, we further conducted a human study. In our human study, we only compare CCGRA with CodeT5, which can achieve the best performance in the all baselines. We follow the methodology used by Wei et al. [16] and Yang et al. [32] to conduct the human evaluation from three aspects:

- `Similarity` evaluates the semantic similarity between the generated comments and the reference comments.
- `Naturalness` evaluates the fluency of the generated comments.
- `Informativeness` evaluates the amount of content transferred from the code to the generated comments.

We invite five master students, who have 1∼3 years of smart contract experience and have good English reading ability. Due to the high cost of manually analyzing all these samples in the testing set, we use a commonly-used sampling method [33] to select the minimum random samples. The number of selected samples can be determined via the following formula:

$$MIN = \frac{n_0}{1 + \frac{n_0 - 1}{size}} \tag{5}$$

where n_0 is related to the confidence level and the error margin $n_0 \left(= \frac{Z^2 \times 0.25}{e^2}\right)$. Here Z is the confidence level score and e is the error margin. $size$ is the size of the testing set. In this RQ, we select MIN examples with the error margin e = 0.05 at 95% confidence level. Specifically, we randomly selected 340 samples from the corpus.

For each code snippet, we generate a questionnaire for each participant. Each participant is asked to score each comment in terms of similarity, naturalness, and informativeness aspects for two comments generated by CCGRA and the baseline CodeT5 respectively. All these scores are integers, ranging from 0 to 4 (the higher the better). During the comment quality evaluation process, the participants can search the Internet for relevant information and unfamiliar concepts. To guarantee a fair comparison, the participants do not know which comment is generated by which approach, and the order of questionnaires is different for different participants. To guarantee the comment evaluation quality, we need each participant to review only 50 code snippets in half a day to avoid fatigue.

TABLE V
RESULTS OF OUR HUMAN STUDY IN TERMS OF SIMILARITY, NATURALNESS, AND INFORMATIVENESS

Approach	Similarity	Naturalness	Informativeness
CodeT5	2.05	3.15	2.15
CCGRA	**2.75**	**3.44**	**2.32**

We compute the average score of the participants' feedback and the results are shown in Table V. In this table, we can find that CCGRA can outperform the approach CodeT5 by 0.70, 0.29, and 0.17 respectively in terms of similarity, naturalness, and informativeness. Therefore, our human study can further verify the competitiveness of CCGRA.

Summary for RQ3: Our human study shows that CCGRA can generate higher quality comments in terms of similarity, naturalness, and informativeness.

VI. THREATS TO VALIDITY

Internal threats. The internal threat is the potential defects in the implementation of our proposed approach and baselines. To alleviate this threat, we first check the code carefully and re-implement baselines according to the original studies.

External threats. The external threat is the choice of corpora. To alleviate this threat, we select the popular corpora, which have been widely used in previous studies on smart contract code summarization [21].

Construct threats. This threat relates to the suitability of our selected performance measures. To alleviate this threat, we consider the widely used performance measures and also conduct a human study to verify the effectiveness of our proposed approach.

VII. CONCLUSION

In this study, we propose a novel retrieval-enhanced approach CCGRA for generating high-quality comments for user-defined code in smart contracts. By leveraging retrieval techniques and pre-trained language models, CCGRA is able to produce reliable and informative comments that improve the comprehensibility and trust-building of smart contract code. We have demonstrated the effectiveness of our approach through extensive experiments and comprehensive analysis. In

addition, a human study was conducted to show that the quality of comments generated by CCGRA outperforms baselines in terms of similarity, naturalness, and informativeness.

In the future, we aim to further improve the performance of CCGRA by exploring advanced code representation methods. Additionally, we plan to expand our dataset by mining more high-quality data of smart contracts, which will facilitate the practical application of our research in various industries, such as finance and healthcare.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Key Foundation of China grant No.62032016 and No.61832014.

REFERENCES

[1] P. Hegedűs, "Towards analyzing the complexity landscape of solidity based ethereum smart contracts," in *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, 2018, pp. 35–39.

[2] L. Shi, F. Mu, X. Chen, S. Wang, J. Wang, Y. Yang, G. Li, X. Xia, and Q. Wang, "Are we building on the rock? on the importance of data preprocessing for code summarization," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 107–119.

[3] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural network." in *IJCAI*, 2020, pp. 3283–3290.

[4] S. Haiduc, J. Aponte, and A. Marcus, "Supporting program comprehension with source code summarization," in *Proceedings of 2010 ACM/IEEE 32nd international conference on software engineering*, vol. 2. IEEE, 2010, pp. 223–226.

[5] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," in *Proceedings of 2010 17th Working Conference on Reverse Engineering*. IEEE, 2010, pp. 35–44.

[6] P. Rodeghero, C. Liu, P. W. McBurney, and C. McMillan, "An eye-tracking study of java programmers and application to source code summarization," *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1038–1054, 2015.

[7] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proceedings of 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 2018, pp. 200–20 010.

[8] Hu, Xing and Li, Ge and Xia, Xin and Lo, David and Jin, Zhi, "Deep code comment generation with hybrid lexical and syntactical information," *Empirical Software Engineering*, vol. 25, no. 3, pp. 2179–2217, 2020.

[9] W. Ye, R. Xie, J. Zhang, T. Hu, X. Wang, and S. Zhang, "Leveraging code generation to improve code retrieval and summarization via dual learning," in *Proceedings of The Web Conference 2020*, 2020, pp. 2309–2319.

[10] G. Yang, X. Chen, Y. Zhou, and C. Yu, "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, 2022, pp. 361–372.

[11] 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.

[12] 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 *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, 2020, pp. 1536–1547.

[13] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "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*, 2021, pp. 2655–2668.

[14] 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.

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

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

[17] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 67–82.

[18] Y. Zhang, S. Ma, J. Li, K. Li, S. Nepal, and D. Gu, "Smartshield: Automatic smart contract protection made easy," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 23–34.

[19] C. Shi, Y. Xiang, J. Yu, K. Sood, and L. Gao, "Machine translation-based fine-grained comments generation for solidity smart contracts," *Information and Software Technology*, vol. 153, p. 107065, 2023.

[20] X. Hu, Z. Gao, X. Xia, D. Lo, and X. Yang, "Automating user notice generation for smart contract functions," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 5–17.

[21] 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.

[22] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *arXiv preprint arXiv:2203.02155*, 2022.

[23] 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.

[24] 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.

[25] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[26] 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.

[27] D. Gros, H. Sezhiyan, P. Devanbu, and Z. Yu, "Code to comment "translation": Data, metrics, baselining & evaluation," in *Proceedings of 2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 746–757.

[28] S. Robertson and H. Zaragoza, *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc, 2009.

[29] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: how far are we?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 373–384.

[30] 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.

[31] S. Stapleton, Y. Gambhir, A. LeClair, Z. Eberhart, W. Weimer, K. Leach, and Y. Huang, "A human study of comprehension and code summarization," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 2–13.

[32] 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.

[33] R. Singh and N. S. Mangat, *Elements of survey sampling*. Springer Science & Business Media, 2013, vol. 15.