International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 03 March 2025
Page No: 740-745
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0103

A Survey on Transformer-based Models in Code Summarization

Suraj Nate[1], Om Patil[2], Shreenidhi Medar[3], Dr. Jyoti Deshmukh[4]
[1,2,3,4]Dept. of AI&DS, Rajiv Gandhi Institute of Tech., Mumbai, Maharashtra, India.
Emails: *surajnate29@gmail.com[1], ompatil9819@gmail.com[2], shreenidhimedar@gmail.com[3], jyoti.deshmukh@mctrgit.ac.in[4].*

Abstract

Code summarization in software engineering refers to the task of automatically generating short natural language descriptions for source code, which has been addressed by the development of transformer-based models such as CodeBERT, CodeT5, and CodeSearchNet in deep learning. These approaches have played a significant role in enhancing the capability of automated code documentation and producing good summaries. This paper provides an in-depth review of the latest transformer-based models for code summarization in terms of architectures, performance metrics, and diverse applications. Through the analysis of the strengths and weaknesses of these models, we aim to give insights that could guide future research and development in the area of automated code documentation. Through this survey, we will evaluate the progress of transformer-based models and give a foundation for further advancements in code summarization technologies.

Keywords: *Code summarization, natural language descriptions, transformer-based models, BERT, CodeT5, CodeBERT, performance metrics, automated code documentation.*

1. Introduction

The field of artificial intelligence (AI) and machine learning (ML) has witnessed a transformative shift with the advent of transformer-based architectures. These models have set new benchmarks across various natural language processing (NLP) tasks, including machine translation, text summarization, and question answering [1]. A significant application of these advancements is in the realm of automated code summarization, a task of generating concise natural language descriptions of source code. This is critical in modern software engineering for improving code readability, maintainability, and developer productivity. Traditional methods for code summarization relied on rule-based or statistical approaches, which were limited in their ability to capture the complex semantics and syntax of programming languages [2]. The introduction of transformers by Vaswani et al. (2017) revolutionized the field by addressing these limitations through the self-attention mechanism, which enables efficient parallel processing and captures context more effectively [4]. Building on this foundation, transformer-based models such as CodeSearchNet [5], CodeT5 [6], and CodeBERT [7] have been adapted to understand and generate code. This survey aims to provide a comprehensive review of transformer-based models in code summarization. We explore their architectures, datasets, evaluation metrics, and practical applications. By analysing their strengths and limitations, we highlight areas for future research and innovation, contributing to the development of robust and efficient automated code documentation tools.

2. Need of Transformer Based Model

Traditional methods to code summarization such as rule-based approaches or classical machine learning models fail to grasp the complex syntax and semantics which are inherent in programming languages [5]. This is mainly because of the following reasons:

- **Semantic Context:** There exist some dynamic behaviors of programming languages including variable dependencies and control flows that could be able to properly interpret the functionality of the code. These kinds of subtle connections are usually omitted from classical models. Transformer models leverage this by learning both local and global contexts in the code, by which they can create meaningful semantically summaries that seize the underlying logic and purpose of the code [6].

- **Scalability:** Traditional models often fail to generalize when applied to diverse programming languages or large-scale datasets. The transformers' ability to pretrain on massive multilingual and multimodal datasets enables them to adapt across different programming contexts, improving scalability and performance for varied summarization tasks [7].
- **Cross-modal Challenges:** Code summarization presents unique challenges as it requires connecting the structured syntax of programming code with the natural language used in summaries. Transformers are particularly effective in this area because they can encode both types of information at the same time, leading to strong cross-modal representations that produce high-quality, human-readable summaries [8].

3. Transformer Models in Code Summarization

The Transformer architecture was proposed by Vaswani et al in 2017. It overcame a number of limitations of seq2seq models due to the introduction of self-attention that captures relationships between distant tokens quite efficiently. Some key transformer-based approaches for code summarization are as follows:

- **CodeBERT:** In 2020, Feng et al. extended the BERT architecture to include both programming and natural languages. Pre-trained on large-scale code datasets, CodeBERT produced tremendous improvements over creating accurate, context-aware summaries. As illustrated in Table 1, CodeBERT produced outstanding performance metrics, such as a BLEU score of 75.4, ROUGE-L of 83.2, and Exact Match (EM) of 61.7, proving its efficiency. CodeBERT extends BERT for programming languages by training on a bimodal dataset consisting of natural language and source code [4]. Pretraining includes two primary objectives: MLM, where the model learns to predict random tokens masked in the input, and replaced token detection, where the model determines whether a token has been replaced by a random token [4]. The bimodal data pair is natural language with its code, like function names and their corresponding implementations, which enables the model to learn cross-modal representations. [21-27]
- **Performance Metric:** CodeBERT has been evaluated using several standard benchmarks and metrics. Below is a summary of its performance:

Table 1 Performance Metric of CodeBERT

Metric	Score
BLEU	75.4
ROUGE-L	83.2
Exact Match (EM)	61.7

- **Limitation of CodeBERT Model:** Despite its effectiveness, CodeBERT has certain limitations. It relies heavily on token-level representations, which can overlook the hierarchical and structural nuances inherent in source code. The model does not explicitly account for syntax trees or data flows, which are crucial for understanding complex code relationships. Furthermore, its performance can be biased toward programming languages and datasets that are overrepresented during pretraining, limiting its generalizability across underrepresented languages and domains. Lastly, the computational overhead of pretraining and fine-tuning large transformer models like CodeBERT is a significant challenge [4]. In fine-tuning, CodeBERT is adapted to specific tasks such as code summarization using supervised datasets annotated with code-text pairs. The self-attention mechanism captures both token-level and semantic dependencies, allowing the model to generate accurate and contextually relevant summaries. [28]

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 03 March 2025
Page No: 740-745
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0103

- **CodeT5:** CodeT5 is the adaptation of T5, that is, Text-to-Text Transfer Transformer to programming tasks [7]. CodeT5 relies on a unified framework for code-related tasks by using span denoising and dual-generation objectives for pretrained models [7]. This helps CodeT5 efficiently handle a wide range of code understanding and generation tasks. As illustrated by Table 2, CodeT5 demonstrated drastic performance improvements for BLEU: 79.1, ROUGE-L: 87.4, and CodeBLEU: 81.33 to demonstrate its proficiency in generating well-informed code summaries that contain accurate context. The span denoising objective masks contiguous spans of tokens within the input sequence and asks it to be predicted; the dual-generation objective combines both code-to-text and text-to-code objectives. These tasks will enhance the capacity of the model to grasp the concept of bidirectionality associated with programming language tasks. CodeT5 would be fine-tuned on a task-specific dataset for code summarization and related tasks so that it could make use of the syntactic and semantic information that has been encoded during pretraining because of its transformer-based encoder-decoder architecture to generate good-quality summaries.[29]
- **Performance Metric**: The performance of CodeT5 is evaluated on retrieval tasks, with the following results:

Table 2 Performance Metric of CodeT5

Metric	Score
BLEU	79.1
ROUGE-L	87.4
CodeBLEU	81.33

- **Limitations of CodeT5 Model:** While CodeT5 achieves state-of-the-art results, it has certain limitations. The model's computational requirements are significant, making it resource-intensive for both pretraining and fine-tuning. Additionally, its reliance on transformer-based architecture limits its ability to effectively handle extremely long input sequences, which are common in real-world programming scenarios. The bidirectional pretraining objectives also introduce challenges when applied to highly domain-specific programming tasks that may require additional fine-tuning or domain adaptation [7].
- **Code Search Net:** CodeSearchNet is a dataset and model for the task of semantic code search, which is finding relevant code snippets based on natural language queries. The model has been pre-trained on a diverse set of six programming languages. Pre-training emphasizes representation learning, which is important both for natural language processing and source code understanding.[9] Contrastive learning is used when fine-tuning the model in this phase, where it can learn to encode code snippets based on their appropriate textual descriptions along with unrelated pairs. It keeps improving its relevance in matching only the relevant information. The difference between these is what helps train the model more effectively in relating the semantics behind code and natural language. As evident from Table 3, the performance of CodeSearchNet is determined with respect to various evaluation metrics, namely, BLEU, ROUGE-L, and CodeBLEU, which shows that it is an efficient model for semantic code search. The architecture for CodeSearchNet in particular - fine-tuning employs the bi-encoder model. This kind of setup can be achieved through two totally distinct encoders; one specific to code snippets and the other towards natural language queries. Therefore, utilizing these two differing types of encoders, it is possible that the model to obtain correctly that code snippet suitable for any particular query. For this reason, the above strategy enhances the effectiveness of the approach for semantic code search tasks toward better returning right and useful outputs. [30]
- **Performance Metric:** The performance of CodeSearchNet is evaluated on retrieval tasks, with the following results:

Table 3 Performance Metric Code Search Net

Metric	Score
BLEU	65.3
ROUGE-L	67.8
CodeBLEU	69.2

4. Limitations of CodeSearchNet Model

CodeSearchNet is incredibly efficient and robust in retrieving the right code snippet for the job. However, this model has its own limitations. The model was designed for use in code-to-text and text-to-code retrieval tasks. That makes it rather unsuitable for use in generative tasks, where a description is produced to summarize what a code block does or why it works a certain way. The model does not make full use of cross-modal representations in such diverse applications. Additionally, CodeSearchNet's performance is closely tied to the languages and datasets it has been pretrained on, which poses challenges for underrepresented programming languages and niche domains. Unlike models like GraphCodeBERT, it does not incorporate structural information such as Abstract Syntax Trees (ASTs), reducing its ability to understand and retrieve code snippets based on deeper code relationships and functionality. It further depends on the quality of the dataset used; hence, it is sensitive to noise, imbalance, or biases, which may deteriorate retrieval accuracy. Lastly, although efficient for smaller-scale tasks, scalability issues remain in the model when handling large repositories or real-time applications, pointing out a need for architectural optimizations to overcome such challenges.

5. Discussion

The recent transformer-based models have significantly improved the field of automated code summarization and become an essential part of modern software development workflows. CodeBERT, CodeT5, and CodeSearchNet all rely on different strengths when handling code-related tasks: CodeBERT in cross-modal understanding, CodeT5 in flexibility in multi-tasking scenarios, and CodeSearchNet in retrieval-based applications. However, high computational costs, biases in the dataset, and scalability concerns remain significant issues in these models. Thus, structural information was incorporated as an example, particularly with the recent GraphCodeBERT models. Their work was in demonstrating syntactic and semantic contexts in code to enhance code representations. Therefore, future studies must improve generalization capabilities in cross-variety programming languages and minimize computation complexity and implement multimodal learning in the incorporation process for accuracy refinement of code summarization. In turn, these issues will continue to drive transformer-based models toward new efficiencies and solutions for automated code documentation and retrieval. [18-20]

Conclusion

Transformer-based models have greatly advanced the field of automated code summarization by challenging traditional strategies. Such models provide summaries that can be human-like for any set of codebases, enhance code comprehension, and improve developer productivity. However, some open issues exist: extra computational overhead, much lower scalability for low-resource environments, and a potential lack of generalizability across a variety of programming languages. Further research should lean toward lightweight architectures with efficiency, multimodal data integration, and advanced structural modeling. The transformers are a good candidate to rewrite the rules of software development and maintenance, therefore becoming an important part of today's programming practices.

Acknowledgements

We give thanks to all the researchers and developers whose work served as a foundation for the development of progressions in transformer-based models and their application in code summarization. Their works in the fields of natural language processing and software engineering have paved the way for this survey. Finally, we are thankful to the open-source community that has given us the datasets, tools, and frameworks for us to experiment and build upon these techniques. We acknowledge the academic support that institutions, peers, and mentors have given us through their guidance and constructive criticism throughout the course of this research. [10-17

References

[1]. Allamanis, M., Barr, E. T., Bird, C., & Sutton, C. (2018). A survey of machine learning for big code and naturalness. ACM Computing Surveys (CSUR), 51(4), 1-37.

[2]. Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. Advances in Neural Information Processing Systems, 27, 3104-3112.

[3]. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30, 5998-6008.

[4]. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[5]. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., & Zhou, M. (2020). CodeBERT: A pre-trained model for programming and natural languages. Findings of EMNLP 2020, 1536-1547.

[6]. Husain, H., Wu, H., Gazit, T., Allamanis, M., & Brockschmidt, M. (2019). CodeSearchNet challenge: Evaluating the state of semantic code search. arXiv preprint arXiv:1909.09436.

[7]. Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Duan, N., & Zhou, M. (2020). GraphCodeBERT: Pre-training code representations with data flow. Findings of ACL 2021, 1294-1306.

[8]. Ahmad, W., Chakraborty, S., Ray, B., & Chang, K. (2021). PLBART: Unified pre-trained model for programming and natural languages. In Findings of ACL 2021.

[9]. Wang, Y., Liu, C., Tang, D., Duan, N., Zhang, M., & Feng, Z. (2021). CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In Proceedings of EMNLP 2021.

[10]. Ahmad, W. U., & Chakraborty, S. (2022). Domain adaptation in pre-trained models for code-related tasks. Proceedings of NAACL 2022, 452–462.

[11]. Zhang, D., Liu, H., & Li, Y. (2022). Enhancing code summarization with static and dynamic analysis. ACM Transactions on Software Engineering and Methodology, 31(4), 1-25.

[12]. Zhang, Y., Yu, L., Liu, Y., & Wang, Q. (2021). Fine-tuning transformer models for code summarization tasks. Journal of Artificial Intelligence Research, 70, 489-507.

[13]. Zhang, L., & Zhou, M. (2022). Multimodal approaches for code summarization: Combining code, comments, and execution traces. Proceedings of the IEEE International Conference on Software Engineering, 978-987.

[14]. Brown, T., Mann, B., Ryder, N., et al., “Language Models are Few-Shot Learners,” in Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901, 2020.

[15]. Barone, J. T., & Sennrich, R. (2017). A Parallel Corpus of Python Functions and Documentation Strings for Automated Code Summarization. Proceedings of EMNLP 2017, 40–45.

[16]. Alon, U., Brody, S., Levy, O., & Yahav, E. (2019). code2vec: Learning Distributed Representations of Code. Proceedings of PLDI 2019, 40–49.

[17]. Iyer, S., Konstas, I., Cheung, A., & Zettlemoyer, L. (2016). Summarizing Source Code using a Neural Attention Model. Proceedings of ACL 2016, 2073–2083.

[18]. Tufano, M., Watson, C., Bavota, G., & Poshyvanyk, D. (2018). Deep Learning Similarities from Different Representations of Source Code. Proceedings of ICSE 2018, 425–435.

[19]. Ahmad, W. U., Chakraborty, S., Ray, B., & Chang, K.-W. (2021). Unified Pre-training for Program Understanding and Generation.

Proceedings of NAACL 2021, 2655–2668.
[20]. LeClair, A., McMillan, C., & Linares-Vásquez, M. (2020). Neural Code Summarization: A Survey of the State of the Art. arXiv preprint arXiv:2004.05688.
[21]. Raffel, C., Shazeer, N., Roberts, A., et al., “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,” JMLR, vol. 21, pp. 1–67, 2020.
[22]. Bansal, T., Sharma, T., & Goel, C. (2021). Improving Code-to-Code Translation Using Transformers. Proceedings of AAAI 2021, 7120–7127.
[23]. Pei, J., Wang, W., Tang, J., & Yu, H. (2021). Fine-Tuning Pretrained Transformer Models for Code Refactoring and Summarization. Proceedings of ASE 2021, 650–659.
[24]. Jain, M., & Jain, S. (2022). An Improved Pretraining Strategy for Multimodal Transformer Models in Code Summarization. Proceedings of IEEE BigData 2022, 2385–2392.
[25]. Kanade, A., Maniatis, P., Balog, G., & Novak, E. (2021). Pre-trained Models for Code: GPT-Coder and Its Applications. arXiv preprint arXiv:2102.04664.
[26]. Das, S., Mukherjee, A., & Chakrabarti, S. (2020). Improving Code Summarization with Edit-Based Attention Mechanisms. Proceedings of COLING 2020, 3401–3413.
[27]. Li, S., Chen, Z., Yin, H., & Yu, Y. (2023). Attention-guided pre-training for code summarization tasks. Proceedings of ICSE 2023, 1505–1515.
[28]. Zhou, J., & Li, X. (2023). Graph-enhanced transformers for semantic code summarization. Proceedings of AAAI 2023, 3460–3471.
[29]. Lin, C., & Sun, M. (2023). Leveraging runtime traces for dynamic code summarization. Proceedings of PLDI 2023, 985–995.
[30]. Hossain, M., & Rashid, M. (2023). Exploring pre-trained multilingual code models for low-resource programming languages. Proceedings of ACL 2023, 1356–1366.