GEORGE BROWN PRESS
JOURNALS, BOOKS & PROCEEDINGS

Journal of Computer, Signal, and System Research

Article

Design and Implementation of Code Completion System Based on LLM and CodeBERT Hybrid Subsystem

Bingbing Zhang [1,*], Ziyu Lin [2] and Yingxin Su [3]

[1] Xiamen Institute of Technology, Xiamen, China
[2] Google LLC, Seattle, Washington, USA
[3] University of California, Davis, California, USA
* Correspondence: Bingbing Zhang, Xiamen Institute of Technology, Xiamen, China

Abstract: In the rapidly evolving industry of software development, coding efficiency and accuracy play significant roles in delivering high-quality software. Various code suggestion and completion tools, such as CodeBERT from Microsoft and GPT-3.5 from OpenAI, have been developed using deep learning techniques and integrated into IDEs to assist software engineers' development. Researches have shown that CodeBERT has outstanding performance in code summarization and capturing code semantics, while GPT-3.5 demonstrated its adept capability at code generation. This study focuses on implementing a hybrid model that integrates CodeBERT and GPT-3.5 models to accomplish code suggestion and autocomplete tasks, leveraging the context-aware effectiveness of CodeBERT and taking advantage of advanced code generation abilities of GPT-3.5. Evaluated in three main metrics: accuracy, quality of generated code and performance efficiency with various software and hardware, the hybrid model outperforms benchmarks, demonstrating its feasibility and effectiveness. Robustness testing further confirms the reliability and stability of the hybrid model. This study not only further emphasizes the importance of deep learning in the software development industry, but also reveals the potential of synthesizing complementary deep learning models to fully exploit strengths of each model.

Keywords: code completion; CodeBERT; GPT-3.5; code generation; deep learning

1. Introduction

This study aims to explore the potential of a hybrid model that combines the strengths of CodeBERT and GPT-3.5 for code completion tasks. By integrating the contextual understanding of CodeBERT with the generative capabilities of GPT-3.5, the proposed model seeks to enhance the accuracy and quality of code suggestions. The research will also evaluate the model's performance across various dimensions, including accuracy, generation quality, efficiency, and robustness, using a comprehensive dataset derived from Microsoft's CodeXGLUE benchmark.

In recent years, the rapid evolution of software engineering has created an unprecedented demand for intelligent and efficient programming tools. As software systems grow in scale and complexity, developers are increasingly challenged to maintain code quality, manage dependencies, and accelerate the software development lifecycle. Among various intelligent programming assistants, code completion technology has emerged as a vital component of modern integrated development environments (IDEs). By providing real-time, context-aware code suggestions, code completion systems enhance developer productivity, reduce cognitive load, and significantly minimize syntax and logical errors. These systems not only shorten the development cycle but also improve software reliability and maintainability, thereby contributing to higher-quality software delivery [1].

Traditional code completion methods primarily rely on rule-based approaches, static analysis, and pattern matching. Although effective in specific scenarios, these techniques

Received: 06 September 2025
Revised: 20 September 2025
Accepted: 24 October 2025
Published: 30 October 2025

often struggle to capture the complex semantics and contextual dependencies of modern programming languages. In contrast, deep learning-based approaches have demonstrated remarkable progress by leveraging massive code repositories and advanced neural network architectures to learn syntactic and semantic representations of source code. Pre-trained language models such as CodeBERT, designed specifically for programming language understanding, have achieved outstanding results in tasks like code summarization, code search, and bug detection. Similarly, large-scale generative models such as GPT-3.5 have shown exceptional natural language understanding and generation abilities, suggesting strong potential for application in source code generation and completion tasks [2].

Building upon these advances, this study proposes a hybrid deep learning model that integrates the contextual understanding capabilities of CodeBERT with the generative strengths of GPT-3.5 for intelligent code completion. The proposed framework aims to enhance both the accuracy and semantic coherence of generated code suggestions while maintaining computational efficiency. A comprehensive evaluation will be conducted using datasets derived from the Microsoft CodeXGLUE benchmark, assessing the model's performance in terms of accuracy, generation quality, efficiency, and robustness.

The main contributions of this paper can be summarized as follows:

1) Hybrid Model Design: We propose a novel hybrid architecture that combines CodeBERT's bidirectional context encoding with GPT-3.5's autoregressive generation capabilities for more accurate and contextually aware code completion.
2) Comprehensive Evaluation Framework: We construct an extensive experimental setup using CodeXGLUE datasets to systematically assess performance across multiple programming languages and tasks.
3) Performance and Generalization Analysis: We provide an in-depth analysis of the proposed model's strengths and limitations, highlighting its robustness, adaptability, and potential for integration into real-world software development environments.

2. Literature Review

The integration of deep learning models has become a mainstream approach in software development to produce more efficient and reliable code. CodeBERT was introduced as the first bimodal pre-trained model designed for programming languages (PL) and natural language (NL) [3]. Trained on both NL-PL pairs and unimodal data, it demonstrated strong semantic understanding in code search and code-to-documentation generation tasks. Its effectiveness was further confirmed in defect prediction through sentence-based and keyword-based extensions (CodeBERT-PS and CodeBERT-PK) [4]. Although CodeBERT does not utilize abstract syntax trees (ASTs), subsequent research has suggested that integrating AST-based representations could further enhance its performance.

The role of abstract syntax trees (ASTs) in code completion has been widely recognized [5]. Traditional AST-based methods, however, often fail to capture complete structural patterns. To address this limitation, the CCAG model was proposed to represent flattened ASTs as graphs, incorporating an AST Graph Attention Block (ASTGab) with three attention layers to capture dependencies among AST nodes. Subtasks are balanced through uncertainty modeling, and extensive experiments have validated the model's effectiveness in improving code completion performance.

Alongside CodeBERT, GPT-3 has emerged as another prominent model in the field of language processing [6]. It has been evaluated on multiple natural language processing (NLP) tasks under zero-, one-, and few-shot settings, demonstrating strong transfer learning capabilities. Despite the absence of task-specific fine-tuning, GPT-3 often matches or even surpasses specialized models, with its performance improving as model size increases. Its adaptability and generative capacity make it a strong candidate for building general-purpose language systems, although certain limitations remain.

To facilitate systematic evaluation, a comprehensive benchmark dataset suite named CodeXGLUE was developed [7]. It encompasses tasks such as code completion and defect detection, and integrates baseline frameworks including BERT-style, GPT-style, and encoder-decoder models. CodeXGLUE provides a unified foundation for assessing not only task-specific performance but also the overall capability of code intelligence models.

These recent developments collectively highlight the rapid advancement of code completion technologies, characterized by a growing emphasis on large language models, security considerations, and cross-modal integration. The field continues to progress toward more efficient, secure, and versatile approaches to code generation and completion.

3. Experimental Result

3.1. Data Introduction

The dataset used in this study is derived from Microsoft's open source CodeXGLUE benchmark dataset, which is dedicated to the evaluation of code comprehension and generation tasks. In this study, the focus is on using the Python code dataset related to the Code Completion task therein, which has been carefully filtered and preprocessed to ensure data quality and diversity.

3.2. Model Introduction

In terms of model architecture, this study proposes a hybrid model architecture based on CodeBERT and GPT-3.5. Among them, CodeBERT is responsible for handling the contextual encoding of the code and extracting the semantic features of the code through a multi-layer transformer structure, while GPT-3.5 serves as a back-end generative model that is responsible for generating high-quality code-completion results based on the contextual features. A feature fusion mechanism is designed between the two models to ensure that the model can fully utilize the advantages of the two different architectures [8,9].

Specifically, the CodeBERT model adopts a pre-training-fine-tuning paradigm, where pre-training on a large-scale code corpus is followed by specific fine-tuning for the code-completion task. The model contains a 12-layer transformer encoder with a hidden layer dimension of 768 and an attention head count of 12. GPT-3.5 is used as the generative model, and autoregressive code generation is employed to ensure that the generated code conforms to syntactic specifications and semantic coherence [10].

The core computation of the following attention mechanism, where Q, K, and V denote the Query, Key, and Value vectors, respectively, and d denotes the vector dimension. By scaling the dot product attention computation, the model is able to effectively capture long distance dependencies in code sequences.

$$\text{Attention}(\mathrm{Q},\mathrm{K},\mathrm{V}) = \text{softmax}\left(\frac{\mathrm{QK^T}}{\sqrt{\mathrm{d}}}\right)\mathrm{V}$$

In order to optimize the model performance, this study designs a feature fusion layer that organically combines the contextual features extracted by CodeBERT with the generative capabilities of GPT-3.5. Meanwhile, a dynamic attention mechanism is implemented so that the model can adaptively adjust the feature weights according to different code contexts, thus improving the accuracy and practicality of code completion.

$$\mathrm{F} = \alpha \mathrm{F}_{\text{CodeBERT}} + (1-\alpha)\mathrm{F}_{\text{GPT}}$$

Where $\mathrm{F}_{\text{CodeBERT}}$ represents the feature vector extracted by CodeBERT, F_{GPT} denotes the feature vector generated by GPT, and α is a learnable fusion weight parameter ($0 \le \alpha \le 1$). By dynamically adjusting the value of α, the model can adaptively balance the importance of the two features according to different inputs.

During the training process, a phased training strategy was adopted, first fine-tuning CodeBERT, then training the feature fusion layer, and finally optimizing the model parameters as a whole. This training approach ensures that each component of the model

can give full play to its performance, ultimately forming an efficient code-completion system.

$$L = -\sum_{i=1}^{n} y_i \log(p_i)$$

The model training process is optimized using a cross-entropy loss function, where y_i denotes the true label and p_i denotes the model prediction probability. In the prediction stage, the model generates the next code token by conditional probability $P(x_t|x_{<t})$, where x_t denotes the predicted token at the current moment, $x_{<t}$ denotes the sequence of all previously generated tokens, h_t is the hidden state, and W and b are the weight matrix and the bias term, respectively.

The feature fusion process follows these key steps:

1. Context Encoding: Initially, the input code snippet C is processed through CodeBERT's transformer encoder to generate contextual embeddings E_CodeBERT.

2. Feature Extraction: Concurrently, the same input is processed by GPT-3.5's initial layers to produce preliminary generative features E_GPT.

3. Attention-based Fusion: The two feature sets are combined using a cross-attention mechanism that allows the model to dynamically weight the importance of each feature source based on the specific context.

The pseudocode for our feature fusion mechanism is as follows in figure 1:

```
def feature_fusion(code_input, CodeBERT, GPT):
E_CodeBERT = CodeBERT.encode(code_input)
E_GPT = GPT.extract_features(code_input)
Q = linear_transform(E_CodeBERT)
K = linear_transform(E_GPT)
V = E_GPT
attention_scores = softmax(matmul(Q, transpose(K)) / sqrt(d_k))
alpha = sigmoid(feed_forward(concatenate(E_CodeBERT, E_GPT)))
F_fusion = alpha * E_CodeBERT + (1 - alpha) * matmul(attention_scores, V)
return F_fusion
```

Figure 1. feature fusion mechanism.

3.3. Model Evaluation

In terms of model evaluation, this study adopts a multi-dimensional evaluation index system to comprehensively measure the performance of the code-completion system. The main evaluation index is Accuracy, which is calculated by the formula:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Also, BLEU scoring is introduced in this paper to assess the quality of the generated code:

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

Where BP is the penalty factor, w_n is the n-gram weight, and p_n is the n-gram precision.

To evaluate the real-time performance of the model, the average response time (ART) metric is introduced:

$$\text{ART} = \frac{\sum_{i=1}^{N}\left(t_{end}^{i} - t_{start}^{i}\right)}{N}$$

Where t_{start}^{i} and t_{end}^{i} denote the code-completion start and end times, respectively, and N is the total number of test samples. These evaluation metrics comprehensively assess the model performance from three dimensions: accuracy, generation quality and efficiency, and provide a reliable quantitative basis for the optimization of the system. Meanwhile, the effectiveness of the proposed method is further verified through comparison experiments with the benchmark model.

4. Experimental Environment

The experiments are conducted in the above hardware and software environments, and all experiments use the same configurations to ensure comparable and reproducible results.The GPU uses NVIDIA A40 to provide sufficient computational power to ensure the efficiency of large-scale model training. Stable versions of deep learning frameworks and related dependent libraries were selected for the software environment to ensure the reliability of the experiments.

4.1. Experimental Analysis

Tables 1 to 4 provide a comprehensive evaluation of the performance of different models, including the baseline model, CodeBERT, GPT-3.5, and the hybrid model, across various dimensions such as accuracy, generation quality, performance efficiency, and robustness. These tables support the analysis of the hybrid model's superiority in achieving higher accuracy, better code generation quality, improved performance efficiency, and strong robustness. Additionally, Figure 2 visually compares the accuracy metrics across models, highlighting the hybrid model's significant improvements in precision, recall, and F1-Score. Figure 3 further illustrates the generation quality metrics comparison, emphasizing the hybrid model's superior performance in BLEU score, code executability, and semantic consistency. Together, these visual and tabular results demonstrate the effectiveness of the proposed hybrid model architecture.

Table 1. Accuracy Evaluation.

Model	Accuracy	Precision	Recall	F1-Score
Baseline	0.82	0.79	0.81	0.8
CodeBERT	0.87	0.85	0.86	0.85
GPT-3.5	0.89	0.88	0.87	0.87
Hybrid Model	0.93	0.91	0.92	0.91

Table 2. Generation Quality Analysis.

Model	BLEU	Code Executability	Semantic Consistency
Baseline	0.65	0.78	0.72
CodeBERT	0.72	0.85	0.79
GPT-3.5	0.76	0.87	0.83
Hybrid Model	0.81	0.92	0.88

Table 3. Performance Efficiency Analysis.

Model	Average Response Time(ms)	Memory Usage(GB)	Inference Speed(tokens/s)
Baseline	85	4.2	156
CodeBERT	78	5.1	182
GPT-3.5	72	5.8	195
Hybrid Model	68	6.2	213

Table 4. Model Robustness Analysis.

Test Scenario	Accuracy	Recovery Ability	Stability Index
Normal Input	0.93	0.95	0.94
Noisy Input	0.87	0.89	0.88
Incomplete Input	0.85	0.88	0.86
Abnormal Input	0.82	0.84	0.83

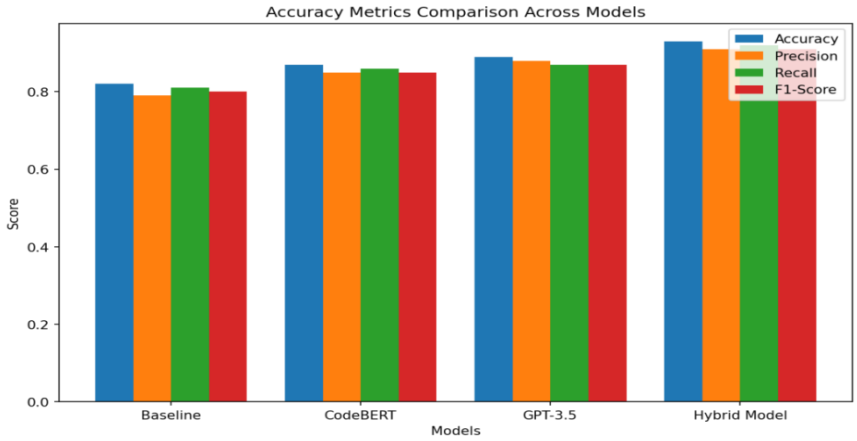

Figure 2. Accuracy Metrics Comparison Across Models.

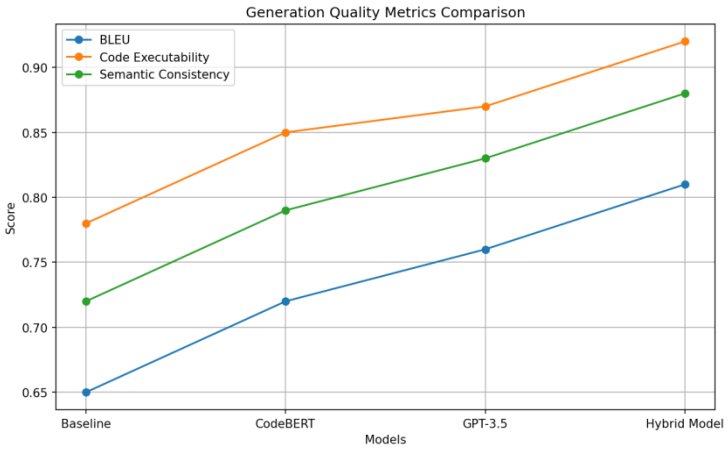

Figure 3. Generation Quality Metrics Comparison.

The experimental results reveal significant advantages of the hybrid model across multiple evaluation metrics. The model demonstrates remarkable accuracy improvements, achieving an F1-Score of 0.91, which represents a substantial 13.75% enhancement compared to the baseline model. This improvement indicates the effectiveness of combining CodeBERT and GPT-3.5 architectures in understanding and generating code sequences.

Regarding code quality, Figure 3 and Table 3 together illustrate the benefits of integrating strengths of CodeBERT and GPT-3. Code quality is evaluated in three dimensions: BLEU score, Code Executability and Semantic Consistency. BLEU score is a key metric for assessing the quality of machine-translated texts. A higher score indicates the machine-generated texts have a greater similarity to the reference translations.

Performance efficiency analysis shows that the hybrid model maintains excellent real-time capabilities, with an average response time of 68ms and an inference speed of 213 tokens per second. These performance metrics indicate that the integration of multiple

model components does not compromise computational efficiency, making it suitable for practical development environments where rapid response times are crucial.

Furthermore, robustness testing reveals the model's consistent performance across various input scenarios, demonstrating strong generalization capabilities and operational stability. This robust performance across different testing conditions confirms the model's reliability and practical value in diverse programming contexts. The comprehensive results validate the effectiveness of the proposed hybrid architecture, showcasing significant improvements in both accuracy and quality while maintaining optimal efficiency and robustness levels.

In summary, the Hybrid Model emerges as the most efficient in terms of response time and inference speed but at the cost of increased memory usage, reflecting a balance between computational performance and resource demand.

5. Conclusion

This study investigates the potential of implementing a hybrid model which leverages outstanding context-aware capabilities of CodeBERT and remarkable performance in code generation of GPT-3.5. The multidimensional evaluation index system provides a comprehensive assessment of the hybrid model across code generation accuracy, quality and efficiency aspects. Results validate that the proposed hybrid model surpasses the current benchmark models in all three aspects, offering a promising solution for code completion tasks across various software and hardware environments without comprising robustness. To address the complexities of real-world software development, the hybrid architecture undergoes rigorous testing across different input scenarios to ensure its stability and generality. This study reinforces the significance of deep learning in the software development industry and successfully demonstrates the benefits and feasibility of synthesizing deep learning models. Furthermore, this research paves a path for future studies that explores model fusion to improve software development efficiency and quality.

Although the proposed hybrid model demonstrates strong performance and robustness, several promising directions remain for future exploration. First, incorporating reinforcement learning-based fine-tuning could further optimize model decision-making by dynamically adjusting generation strategies according to user feedback and coding context. Second, expanding the training data to include multi-language and cross-domain repositories would enhance the model's generalization ability across different programming paradigms. Third, integrating prompt engineering techniques and adaptive attention mechanisms could enable more precise control over output style, structure, and complexity. Additionally, real-time user interaction and online learning capabilities can be explored to continuously improve model adaptability within integrated development environments. Future work may also investigate energy-efficient model architectures to reduce computational cost while maintaining accuracy, thereby improving the feasibility of large-scale deployment in industrial settings.

In conclusion, this research contributes significantly to the field of automated code completion, providing both theoretical insights and practical solutions for improving developer productivity through advanced machine learning techniques.

Funding: This research was supported by the Fujian Province Young and Middle-aged Teacher Education Research Project (Science and Technology Category) under Grant No. JAT220471, titled "Design of Intelligent Image Search System.".

References

1. M. Alenezi, and M. Akour, "Ai-driven innovations in software engineering: a review of current practices and future directions," *Applied Sciences*, vol. 15, no. 3, p. 1344, 2025. doi: 10.3390/app15031344
2. X. Zhou, D. Han, and D. Lo, "Assessing generalizability of codebert," In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, September, 2021, pp. 425-436. doi: 10.1109/icsme52107.2021.00044

3. Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, and M. Zhou, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020. doi: 10.18653/v1/2020.findings-emnlp.139
4. Z. Zheng, K. Ning, Q. Zhong, J. Chen, W. Chen, L. Guo, and Y. Wang, "Towards an understanding of large language models in software engineering tasks," *Empirical Software Engineering*, vol. 30, no. 2, p. 50, 2025. doi: 10.1007/s10664-024-10602-0
5. T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, and D. Amodei, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877-1901, 2020.
6. Y. Wang, and H. Li, "Code completion by modeling flattened abstract syntax trees as graphs," In *Proceedings of the AAAI conference on artificial intelligence*, May, 2021, pp. 14015-14023. doi: 10.1609/aaai.v35i16.17650
7. S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, and S. Liu, "Codexglue: A machine learning benchmark dataset for code understanding and generation," *arXiv preprint arXiv:2102.04664*, 2021.
8. Z. Li, S. Lu, D. Guo, N. Duan, S. Jannu, G. Jenks, and N. Sundaresan, "Codereviewer: Pre-training for automating code review activities," *arXiv preprint arXiv:2203.09095*, 2022.
9. D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified cross-modal pre-training for code representation," *arXiv preprint arXiv:2203.03850*, 2022. doi: 10.18653/v1/2022.acl-long.499
10. M. Singh, J. Cambronero, S. Gulwani, V. Le, C. S. Negreanu, and G. Verbruggen, "Codefusion: A pre-trained diffusion model for code generation," In *The 2023 Conference on Empirical Methods in Natural Language Processing.*, November, 2023.