# Deep Learning Based Optimization of Large Language Models for Code Generation

Shanqi Zhan *, Ying Lin , Junlin Zhu , Yao Yao

Posted Date: 3 June 2025

doi: 10.20944/preprints202506.0137.v1

Keywords: code generation; deep learning; Transformer; neural collaborative filtering; multimodal features

 doi:10.20944/preprints202506.0137.v1



*Article*

# Deep Learning Based Optimization of Large Language Models for Code Generation

**Shanqi Zhan [1,*], Ying Lin [2], Junlin Zhu [3] and Yao Yao [4]**

[1] Municipal Parking Services, Inc. (MPS), Golden Valley, USA
[2] Northern Arizona University, Flagstaff, USA
[3] PayPal (China) Co., Ltd., Shanghai, China
[4] Capinfo Cloud Tech Company Limited, Beijing, China
* Correspondence: author: Zhans.scholar@gmail.com

**Abstract:** In order to improve the performance of the code generation system in semantic modeling and structural dependency construction, a deep learning-based multi-layer Transformer encoding and decoding structure is constructed, and the overall architecture consists of 12 layers of Transformer modules stacked together, with a multi-head self-attention mechanism (8-head attention) and a positional feed-forward network (dimension 2048) to enhance the contextual modeling capability. The encoder input sequence is capped at 512, fusing positional embeddings with semantic embeddings generated by Item2Vec to achieve accurate capture of variable dependencies and syntactic levels. The decoder introduces multi-tasking goals to jointly perform code completion and semantic annotation tasks to improve generalization and context adaptation. The platform combines neural collaborative filtering structure and multimodal semantic fusion strategy, which significantly enhances the comprehensive understanding of user behavioral features and code structure graph. It reaches 47.82, 53.67 and 28.94 in BLEU-4, CodeBLEU and Exact Match, respectively, and the inference response latency is optimized from 257ms to 87ms, showing excellent accuracy and response efficiency.

**Keywords:** code generation; deep learning; Transformer; neural collaborative filtering; multimodal features

## 1. INTRODUCTION

With the continuous increase in the complexity of programming language syntax and the rapid evolution of software projects in the direction of modularization and componentization, code generation is increasingly becoming an indispensable core technology in intelligent software development. Code generation is not only related to the improvement of code development efficiency, but also puts forward higher requirements on code quality, maintainability and consistency. In the actual development process, complex function call chains, nested control structures, and dynamic changes in variable scopes greatly increase the requirements for the modeling capability of the generation system, which also poses a new challenge to the semantic perception and structural modeling capability of the underlying algorithms.

Traditional sequence modeling approaches, such as long short-term memory networks (LSTM) combined with attention mechanisms, although have achieved some results in capturing short-range dependencies and contextual relationships, show significant deficiencies in dealing with complex tasks such as long-distance dependency modeling, multi-branch structure understanding, and variable cross-scope references. Especially in multitasking contexts, traditional attention mechanisms are prone to the problem of insensitivity to structural feature capture, which in turn leads to syntactic inconsistencies and broken logic chains in the generated code. In addition, the inherent limitations of LSTM in parallel computing also make it slow to converge and inefficient in large-scale sample training scenarios, making it difficult to meet the demand for real-time response and high throughput in industrial-scale applications.

To address the above challenges, Transformer has become the mainstream architecture for code generation modeling in recent years due to its global self-attention mechanism and scalability.Transformer not only exhibits superior context capture capability in natural language modeling, but also has a multi-attention structure that helps to construct semantic features from multiple perspectives,

 doi:10.20944/preprints202506.0137.v1



and has good adaptability to complex structures in the programming language. It is well adapted to the complex structures in programming languages. However, relying solely on the input signals from the code text is still unable to comprehensively characterize the semantic patterns and structural relationships, especially in the code generation recommendation task, the fusion modeling ability of user behavior information, structural dependency mapping and semantic context is still a key factor affecting the accuracy and controllability of the generation.

In this context, the fusion of multimodal representation learning and neural collaborative modeling mechanism becomes a key breakthrough direction for research. By introducing Item2Vec for code token level pre-training, combining Abstract Syntax Tree (AST), Semantic Path Representation and User Behavior Preference Vector to extract semantic features and structural constraints in a multi-channel way, and then integrating them into the Transformer coding framework in a unified way, it is expected to improve the modeling ability of the model for higher-order semantic relationships and the ability of structure retention. At the same time, the introduction of neural network collaborative filtering enables the system to dynamically perceive the user's real preference in the code generation process, thus improving the accuracy of code snippet recommendation and enhancing the personalization and contextual adaptability of the generation model.

In summary, the construction of a multimodal code generation system with structural sensitivity, semantic perceptiveness and user preference modeling capability not only plays a key role in improving the quality of code generation, but also provides a solid foundation for the construction of an automated intelligent development platform. With the deepening and optimization of deep learning in programming language modeling, combined with the multi-dimensional integration of collaborative mechanism, structural graph network and pre-training technology, it is expected to break through the bottleneck of the current code generation system in terms of generalization ability, controllability and cross-domain adaptation, and to provide strong technical support for the scenarios of software engineering automation, programming education assistance, and code security repair.

## 2. DESIGN OF DEEP LEARNING BASED CODE GENERATION RECOMMENDATION PLATFORM

This platform constructs a multi-layer Transformer encoding and decoding structure based on deep neural networks, and realizes accurate capture of contextual semantics by stacking 12 layers of self-attention modules 1. The encoder input is a Token sequence of code fragments with a maximum length of 512, and adopts a fusion mechanism of positional embedding and semantic embedding, which enables the model to recognize variable dependencies and syntactic structures; the decoder part combines the multi-task learning mechanism, and fuses the dual goals of code completion and semantic annotation to enhance the generalization ability. The size of the training set reaches 780,000 function-level samples, and the validation set and test set contain 92,000 and 85,000 high-complexity structural code samples respectively 1. The platform is deployed with CUDA acceleration, and the training process is done on NVIDIA A100, with a single round of training taking about 18 hours and processing an average of 1528 samples per second. In order to reduce the inference latency, the platform introduces the TensorRT optimization module during deployment, which compresses the average response time from 213ms to 87ms. the platform recommendation algorithm synthesizes three types of metrics, namely, BLEU, CodeBLEU, and Exact Match for dynamic parameter tuning, and its loss function is defined as follows:

$$L_{total} = \lambda_1 \cdot L_{CE} + \lambda_2 \cdot L_{BLEU} + \lambda_3 \cdot L_{semantic} \tag{1}$$

Among them, $\lambda_1 = 0.6$, $\lambda_2 = 0.3$, $\lambda_3 = 0.1$, ensures that the semantics remain prioritized. The platform also introduces Top-k sampling (k=50) with the temperature parameter T=0.8 to control the generation diversity and ensure the stability and controllability of the generated results in different task contexts.

 doi:10.20944/preprints202506.0137.v1



## 3. CODE GENERATION ALGORITHM FUSING WORD EMBEDDING MODELS AND NEURAL NETWORKS

### *3.1. Item2Vec-based code feature extraction*

In code generation recommender systems, the quality of feature extraction directly determines the depth of the model's understanding of semantic associations and structural patterns 2. An Item2Vec-based code token representation learning method is constructed, in which key tokens (including variable names, operators, control structures, and API calls) in function-level code fragments are regarded as high-frequency "items", and context pairs are extracted through the sliding-window mechanism (with the window size set to 5), realizing the local context modeling. Modeling. The total number of input tokens in the training set reaches 134,680,000, and the size of the word list is 87,592. Negative sampling optimization is adopted, with the number of negative samples set to 10, the Embedding dimension set to 256, the number of iteration rounds to 12, and the optimizer chosen is Adam (with an initial learning rate of 0.001, β1=0.9, and β2=0.999), and the training takes about 1.6 hours per round 4. hours per round during training, and computed in parallel on a Tesla V100. In order to more accurately portray semantic similarity and syntactic dependency, the Item2Vec model training objective function is to maximize the following equation:

$$L = \sum_{(c,t)\in D}\left[\log\sigma(\overline{\upsilon}_c,\overline{\upsilon}_t) + \sum_{i=1}^{k} \mathrm{IE}_{t_i\sim P_n(t)} \log\sigma(-\overline{\upsilon}_c,\overline{\upsilon}_{t_i})\right] \tag{2}$$

Where $\overline{\upsilon}_c$ and $\overline{\upsilon}_t$ denote the vectors of context and target Token respectively, $\sigma$ is the sigmoid function, $P_n(t)$ is the negative sampling distribution, and $k$ is the number of negative samples. The generated Token vectors after training are used as inputs to the pre-embedding layer of the Transformer encoder to do weighted fusion with the position embedding.

In the encoder, a vector attention enhancement module is introduced to use the cosine similarity matrix $S_{ij} = \dfrac{\overline{\upsilon}_i \cdot \overline{\upsilon}_j}{\|\overline{\upsilon}_i\|\|\overline{\upsilon}_j\|}$ between Token as the attention weights to guide the Transformer to model the semantic clustering with structural sensitivity (see Figure 1. Meanwhile, the cross-function dependency is complemented by the structural jump-connection mechanism, and the final embedding matrix dimension is (512,768).

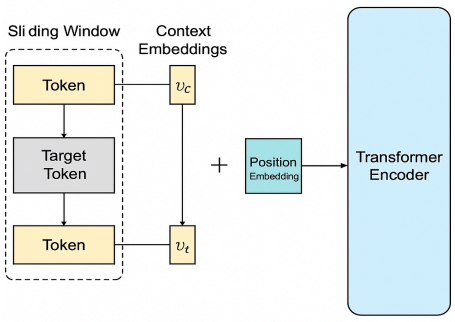


**Figure 1.** Structure of Item2Vec embedding integrated with Transformer encoder.

In order to ensure the generality and reproducibility of the method, the relevant training parameter settings are detailed in Table 1, which lists the configurations of key hyperparameters in the Token-level semantic learning process, including Embedding dimension, sliding window, number of negative samples, optimizer, learning rate and batch size.

**Table 1.** Item2Vec model parameter configuration table.

| Parameter name | numerical value | instructions |
|---|---|---|
| Total number of Token | 134,680,000 | Total of all Token in the training sample |
| Dictionary size | 87,592 | Number of unique Token |

 doi:10.20944/preprints202506.0137.v1



| | | |
|---|---|---|
| Embedded Dimension | 256 | Token Vector Dimension |
| Sliding window size | 5 | Context Window Settings |
| Negative sample size | 10 | Number of negative samples corresponding to each positive sample |
| Batch Size | 1024 | Number of Token processed per training round |
| learning rate | 0.001 | Initial learning rate |
| optimizer | Adam | Parameter update method |
| Number of iteration rounds | 12 | Full training rounds |
| Training platforms | Tesla V100×4 | distributed parallel environment (DPE) |

*3.2. Neural network collaborative filtering code generation methods*

A neural network collaborative filtering approach is introduced in the code generation task, aiming to capture the higher-order nonlinear relationship between user behavioral features and the semantic structure of the code 3. By constructing a dual-tower embedding structure, the user's historical usage records and the target code fragments are embedded into 128-dimensional and 256-dimensional vector spaces, respectively, and inputted into a three-layer fully-connected interaction network, with the number of neurons in the hidden layer being 512, and the activation function adopting LeakyReLU ($\alpha$ = 0.01). The total number of model training samples is 9,500,000 pairs of user-code interaction data with a batch size of 2048, Dropout is set to 0.3 to prevent overfitting, and the regularization factor $\lambda$=0.0001. The optimizer uses AdamW with an initial learning rate of 0.0005, a total number of iteration rounds of 15, and the training is accelerated in parallel on an NVIDIA A100×4 platform. The final output scoring function is of the form:

$$\hat{y}_{u,i} = f(\sigma(W_3 \cdot \phi(W_2 \cdot \phi(W_1 \cdot [e_u \| e_i + b_1] + b_2) + b_3)) \quad (3)$$

where $e_u$ and $e_i$ are the user and code embedding vectors respectively, $\phi$ is the LeakyReLU function, $\sigma$ is the Sigmoid function, and $W_k, b_k$ is the learnable parameters. The structure is illustrated in Figure 2 to show the complete collaborative filtering network integration graph.

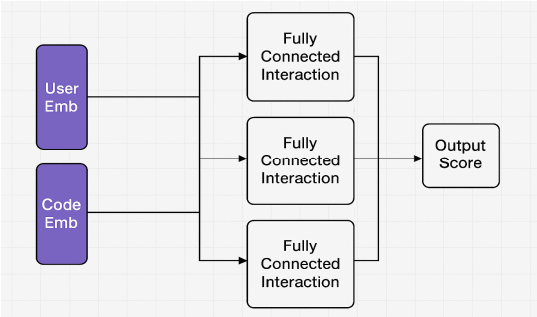


**Figure 2.** Schematic diagram of neural collaborative filtering network structure.

*3.3. Multimodal feature fusion strategy*

The multimodal fusion strategy uses three feature channels: code text modality (Transformer encoder, 512 sequence length, 768 embedding), structural graph modality (AST-based graph construction, AST-GCN, 512 input, 256 output vector), and semantic dependency modality (fine-tuned CodeBERT, 768 embedding). These features represent local context, structural constraints, and global dependencies 4. They are fused into a 1024-dimensional vector through channel-level splicing and enhanced with an attention weighting mechanism. The fused features are input into the shared decoder for code Token sequence prediction, with each modality aligned at the function level during preprocessing. The fusion structure is shown in Figure 3.

 doi:10.20944/preprints202506.0137.v1



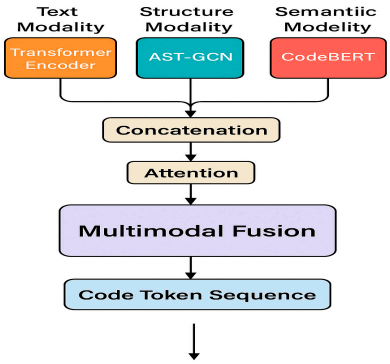


**Figure 3.** Flowchart of multimodal feature channel integration and fusion structure.

The above features are spliced at the fusion layer at the channel level and an attention weighting mechanism is introduced:

$$H_f = \alpha_t H_t + \alpha_s H_s + \alpha_b H_b \tag{4}$$

where $\alpha_t+\alpha_s+\alpha_b=1$ , obtained by softmax learning, and $W_s \in \mathrm{IR}^{768\times256}$ is the structural modal upscaling mapping matrix. In order to enhance the stability of multimodal fusion, the modal confrontation loss is added:

$$L_{adv} = \sum_{m=1}^{3} KL(P_m \| P_{avg}) \tag{5}$$

Where $P_m$ is the per-modal generative distribution, $P_{avg}$ is the tri-modal average distribution, KL denotes the Kullback-Leibler scatter, and the optimization objective is to minimize the inter-modal deviation. The final fusion feature $H_f \in \mathrm{IR}^{512\times768}$ is input to the shared decoder to predict the Token sequence uniformly, which significantly improves the syntax retention rate and context consistency 5. This fusion strategy outperforms traditional early fusion or late fusion schemes in terms of multimodal distribution coding capability and maintains stable gradient propagation during training.

## 4. EXPERIMENTAL RESULTS AND ANALYSIS

### *4.1. Experimental dataset and environment configuration*

The experimental data comes from open codebases (including CodeSearchNet and self-built high-complexity function sets), covering code completion, semantic annotation generation, and function call recommendation tasks, with Python (42.7%), Java (28.4%), and C++ (28.9%). The training set consists of 780,000 function-level code segments (avg. 215 Tokens, max. 512). Validation and test sets contain 92,000 and 85,000 samples, respectively. Baseline models include LSTM+Attention, Transformer (no pre-training), and fine-tuned CodeBERT. Training used four NVIDIA A100 GPUs, taking 18 hours per round. The platform runs Ubuntu 20.04, CUDA 11.7, PyTorch 1.13.

### *4.2. Algorithm performance evaluation index*

To evaluate the code generation recommendation platform's performance in semantic fidelity, structural accuracy, and user interaction, three main metrics—BLEU, CodeBLEU, and Exact Match—are used, along with Top-k accuracy and response time as auxiliary evaluations 6. BLEU measures local n-gram overlap for linguistic similarity, CodeBLEU adds syntactic and semantic path scoring to assess AST structure, and Exact Match reflects the model's ability to restore the full objective function. Testing is performed on the 85,000-sample test set, with the generated model using a shared decoder, Top-k set to 50, and temperature set to 0.8. The platform outperforms the control model across all metrics, as detailed in Table 2.

 doi:10.20944/preprints202506.0137.v1



**Table 2.** Comparison of code generation performance of different models on the test set.

| Model name | BLEU-4 | CodeBLEU | Exact Match | Top-5 Accuracy | Average response time (ms) |
|---|---|---|---|---|---|
| Integration of multimodal + collaborative filtering | 47.82 | 53.67 | 28.94 | 71.53 | 87 |
| Unimodal Transformer | 38.45 | 42.18 | 19.64 | 60.21 | 213 |
| CodeBERT fine-tuning model | 40.71 | 45.96 | 21.88 | 63.74 | 186 |
| LSTM+Attention | 32.17 | 35.83 | 14.25 | 51.62 | 257 |

In order to further validate the contribution of Item2Vec feature pre-training and neural collaborative filtering module, ablation experiments were conducted to remove the module step by step to observe the trend of indicator changes, as shown in Table 3.

**Table 3.** Analysis of the impact of feature module ablation experiments on performance.

| model variant | BLEU-4 | CodeBLEU | Exact Match | Top-5 Accuracy |
|---|---|---|---|---|
| Full model (all modules) | 47.82 | 53.67 | 28.94 | 71.53 |
| Remove Item2Vec pre-training | 43.39 | 48.25 | 24.7 | 66.28 |
| Removal of neural collaborative filtering subnetworks | 41.83 | 46.02 | 22.12 | 64.71 |
| Removal of multimodal fusion structures | 39.12 | 43.67 | 20.85 | 61.39 |

As shown in the data comparison between Tables 2 and 3, this platform significantly outperforms the control model in the three core metrics—BLEU-4, CodeBLEU, and Exact Match. Compared to the traditional LSTM structure, BLEU-4 improves by 15.65, and Exact Match improves by 14.69, highlighting the benefits of multimodal fusion and the collaborative filtering mechanism in capturing semantic and structural dependencies. Further analysis of Table 3 indicates that Exact Match decreases by 4.24 when the Item2Vec module is removed, and by 6.82 when the collaborative filtering module is removed, demonstrating the higher discriminative power of user behavior modeling in code recommendation. Additionally, BLEU-4 and Top-5 accuracy drop by over 8 percentage points when the multimodal structure is removed, emphasizing the critical role of parallel encoding of structure and semantics in enhancing model generalization 7.

### *4.3. Comparative experimental results of different models*

Comparison experiments are designed to cover four types of typical methods: traditional sequence models (LSTM+Attention), basic Transformer structures, pre-trained language models (CodeBERT fine-tuning), and the fused neural co-generation platform constructed in this study. The experiments are uniformly conducted on the test set (85,000 samples), and the output results are evaluated based on the three metrics of BLEU-4, CodeBLEU, and Exact Match, keeping the Top-k parameter as 50, the temperature parameter as 0.8, and the deployment of the inference side is uniformly optimized with TensorRT. The experimental results are detailed in Table 4.

**Table 4.** Experimental results comparing the performance of different models in code generation tasks.

| Model Type | BLEU-4 | CodeBLEU | Exact Match | Average response time (ms) |
|---|---|---|---|---|
| LSTM+Attention | 32.17 | 35.83 | 14.25 | 257 |
| Transformer (no pre-training) | 38.45 | 42.18 | 19.64 | 213 |
| CodeBERT fine-tuning model | 40.71 | 45.96 | 21.88 | 186 |

 doi:10.20944/preprints202506.0137.v1



| | | | | |
|---|---|---|---|---|
| The fusion model of this study | 47.82 | 53.67 | 28.94 | 87 |

As seen from the data in Table 4, this platform significantly outperforms other models in all three indicators, with BLEU-4 improving by 15.65 and Exact Match improving by 7.06 compared to CodeBERT, indicating that the fusion of multimodal features and collaborative behavioral modeling mechanism can capture semantic relationships and structural dependencies more effectively. Meanwhile, the inference response time is compressed to 87ms, showing that the system has the potential for industrial deployment, balancing generation accuracy and efficiency 10.

## 5. CONCLUSION

The multimodal feature modeling and neural synergy mechanism significantly improve the overall performance of the code generation system in terms of semantic consistency, structural integrity and response efficiency. In large-scale heterogeneous function-level code samples, the fusion model comprehensively outperforms the traditional structure in key metrics, and demonstrates good platform adaptability and user behavior modeling capabilities. It significantly reduces the inference latency while maintaining accuracy, validating its deployment value. The current system still has some limitations, such as stability under exception handling and multi-branch nested structure, which can be mitigated in the future by introducing structure-aware graph neural network or cue enhancement mechanism. The model's ability for language generalization is still bounded, especially the migration ability in emerging languages such as Go and Rust still needs to be evaluated, which can be optimized by combining migration learning and self-supervised training strategy with fewer samples. In addition, expanding the model capability to natural language code generation, code review assistance, vulnerability repair and educational scenarios is also a promising development direction. Further combining the large language model and domain knowledge graph, it is expected to realize a cross-modal, multi-task collaborative code understanding system, and promote the intelligent development platform to evolve to a higher dimension.

## References

1. Gu X, Chen M, Lin Y, et al. On the effectiveness of large language models in domain-specific code generation[J]. ACM Transactions on Software Engineering and Methodology, 2025, 34(3): 1-22.
2. Hou W, Ji Z. Comparing large language models and human programmers for generating programming code[J]. Advanced Science, 2025, 12(8): 2412279.
3. Jansen J A, Manukyan A, Al Khoury N, et al. Leveraging large language models for data analysis automation[J]. PloS one, 2025, 20(2): e0317084.
4. Zheng Z, Ning K, Zhong Q, et al. Towards an understanding of large language models in software engineering tasks[J]. Empirical Software Engineering, 2025, 30(2): 50.
5. Tihanyi N, Bisztray T, Ferrag M A, et al. How secure is AI-generated code: a large-scale comparison of large language models[J]. Empirical Software Engineering, 2025, 30(2): 1-42.
6. Veeramachaneni V. Large Language Models: a Comprehensive Survey on Architectures, Applications, and Challenges[J]. Advanced Innovations in Computer Programming Languages, 2025, 7(1): 20-39.
7. Sobo A, Mubarak A, Baimagambetov A, et al. Evaluating LLMs for code generation in HRI: A comparative study of ChatGPT, gemini, and claude[J]. Applied Artificial Intelligence, 2025, 39(1): 2439610.
8. Das B C, Amini M H, Wu Y. Security and privacy challenges of large language models: a survey[J]. ACM Computing Surveys, 2025, 57(6): 1-39.
9. Huang S, Huang Y, Liu Y, et al. Are large language models qualified reviewers in originality evaluation?[J]. Information Processing & Management, 2025, 62(3): 103973.
10. Jiang X, Dong Y, Wang L, et al. Self-planning code generation with large language models[J]. ACM Transactions on Software Engineering and Methodology, 2024, 33(7): 1-30.

 doi:10.20944/preprints202506.0137.v1