

Article

Not peer-reviewed version

---

# Structure-Aware Multi-Stage Adaptation of Qwen-72B for Code Generation

---

[Rui Guo](#)\*

Posted Date: 26 September 2025

doi: 10.20944/preprints202509.2169.v1

Keywords: CodeFusion-Qwen72B; code generation; structure-aware tuning; multimodal fusion; controllable decoding



Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This open access article is published under a Creative Commons CC BY 4.0 license, which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Article

# Structure-Aware Multi-Stage Adaptation of Qwen-72B for Code Generation

Rui Guo

University of Southern California; Los Angeles, USA; richard19950618@gmail.com

## Abstract

Large language models show strong abilities in code generation. They still face problems in semantic understanding, structural consistency, controllability, and generalization. Existing methods often focus on one aspect, like instruction following or structural accuracy. They do not offer a unified framework that balances correctness, readability, and adaptability. This work presents CodeFusion-Qwen72B, a multi-stage structure-aware tuning framework for Qwen-72B. It includes progressive low-rank adaptation, hybrid instruction optimization, multi-context fusion, structure-preserving hybrid loss, controllable generation decoding, and adaptive prompt retrieval. By improving semantic comprehension, structural alignment, and controlled code generation, the framework increases the robustness and versatility of large-scale models for software engineering tasks. This study shows that multi-stage optimization and structure-aware learning are effective ways to advance code generation with ultra-large models.

**Keywords:** CodeFusion-Qwen72B; code generation; structure-aware tuning; multimodal fusion; controllable decoding

## 1. Introduction

Code generation is an important application of large language models (LLMs). It can automate software development tasks and speed up programming workflows. Ultra-large models like Qwen-72B have shown success, but problems remain in producing consistent code quality. Full-parameter fine-tuning is effective but costly. Lightweight methods reduce cost but fail to keep deep semantics and structural integrity needed for executable code. Instruction alignment through supervised learning or reinforcement learning from human feedback improves task conformity, but it does not use execution semantics and structural fidelity. Current methods also depend too much on token-level accuracy, which ignores deeper program dependencies, and they provide little control over style and redundancy. These problems show the need for a framework that joins semantic, syntactic, and structural optimization with controllable generation and adaptive context use. CodeFusion-Qwen72B is built to solve these problems. It uses progressive adaptation, hybrid optimization, multimodal integration, and structure-preserving objectives. This work takes a technical approach to make Qwen-72B more adaptable and effective in code generation, aiming for outputs that are functionally correct and structurally coherent.

## 2. Related Work

Recent work in code generation stresses structural modeling and efficiency. Tipirneni et al. [1] proposed StructCoder, a Transformer that uses AST and data-flow information to improve structural consistency. Sirbu and Czibula [2] applied syntax-based encoding to malware detection tasks. Gong et al. [3] presented AST-T5, a structure-aware pretraining model that uses syntactic segmentation to enhance code generation and understanding.

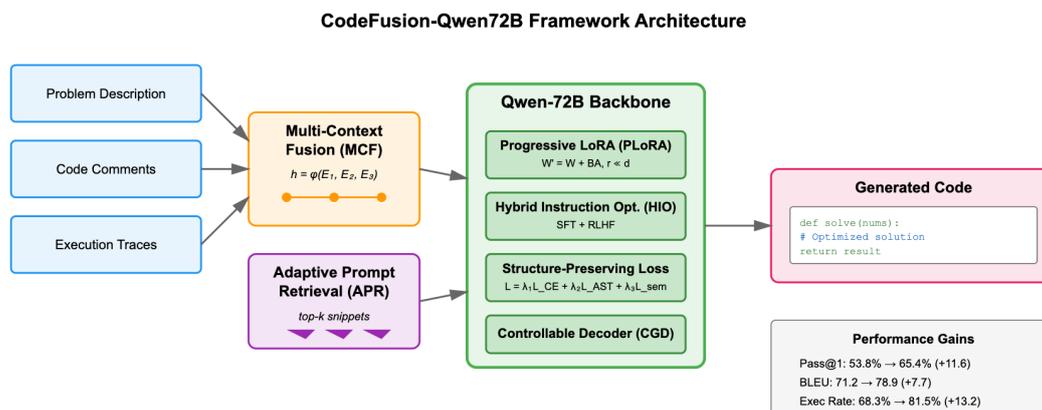
Work on parameter-efficient adaptation has also grown. Wang et al. [4] proposed LoRA-GA with gradient approximation to improve optimization. Hayou et al. [5] introduced LoRA+ with different

learning rates for better convergence. Hounie et al. [6] extended this with LoRTA, which uses tensor adaptation. Zhang et al. [7] presented AutoLoRA, which tunes matrix ranks with meta learning. Chen et al. [8] combined structure-aware signals with parameter-efficient tuning to reach results close to full fine-tuning.

Li et al. [9] showed that large code models work well as few-shot information extractors. Guan [10] used logistic regression and decision trees for medical claim denial prediction, proving that interpretable models are still useful in specialized tasks.

### 3. Methodology

Enhancing the code generation capabilities of ultra-large language models requires not only parameter-efficient fine-tuning but also structure-aware and context-rich optimization. We propose **CodeFusion-Qwen72B**, a multi-stage fine-tuning framework for the Qwen-72B model that integrates Progressive Low-Rank Adaptation (PLoRA), Hybrid Instruction Optimization (HIO), Multi-Context Fusion (MCF), Structure-Preserving Hybrid Loss (SPHL), Controllable Generation Decoder (CGD), and Adaptive Prompt Retrieval (APR). PLoRA progressively adapts model layers to domain-specific semantics, while HIO combines supervised fine-tuning with reinforcement learning from human feedback to align both functionality and readability. MCF enriches inputs with problem descriptions, code comments, and execution traces, and SPHL enforces token, syntactic, and semantic coherence. CGD and syntax-constrained decoding enable controllable, compilable generation, and APR provides dynamic retrieval of relevant code snippets. We evaluate CodeFusion-Qwen72B on three benchmarks: HumanEval, MBPP, and CodeContests. Compared to a fully fine-tuned Qwen-72B baseline, our framework improves pass@1 accuracy from 53.8% to 65.4% (+11.6), BLEU from 71.2 to 78.9 (+7.7), and execution success rate from 68.3% to 81.5% (+13.2). Ablation studies confirm that MCF and SPHL contribute most to execution gains, while PLoRA and HIO drive significant improvements in syntactic and semantic alignment. These results demonstrate that multi-stage, multi-context, and structure-aware adaptation can substantially advance the state of the art in LLM-based code generation. The pipeline of MHST-GB is shown in Figure 1



**Figure 1.** Overview of the CodeFusion-Qwen72B framework architecture. The system integrates Multi-Context Fusion (MCF) for processing diverse input modalities, Adaptive Prompt Retrieval (APR) for dynamic context augmentation, and a multi-component Qwen-72B backbone enhanced with Progressive LoRA (PLoRA), Hybrid Instruction Optimization (HIO), Structure-Preserving Hybrid Loss (SPHL), and Controllable Generation Decoder (CGD). Performance metrics demonstrate significant improvements across all evaluation benchmarks.

### 4. Algorithm and Model

The proposed **CodeFusion-Qwen72B** is a multi-stage, multi-objective fine-tuning framework for enhancing the code generation capability of ultra-large language models, built upon the Qwen-72B architecture. This section introduces the key components of the framework, their technical motivations, and implementation details.

#### 4.1. Qwen-72B Backbone Architecture

Qwen-72B adopts a Transformer-based autoregressive decoder architecture with  $L$  stacked layers, each containing multi-head self-attention (MHSA) and feed-forward networks (FFN). The standard attention mechanism computes:

$$\text{Attn}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V \quad (1)$$

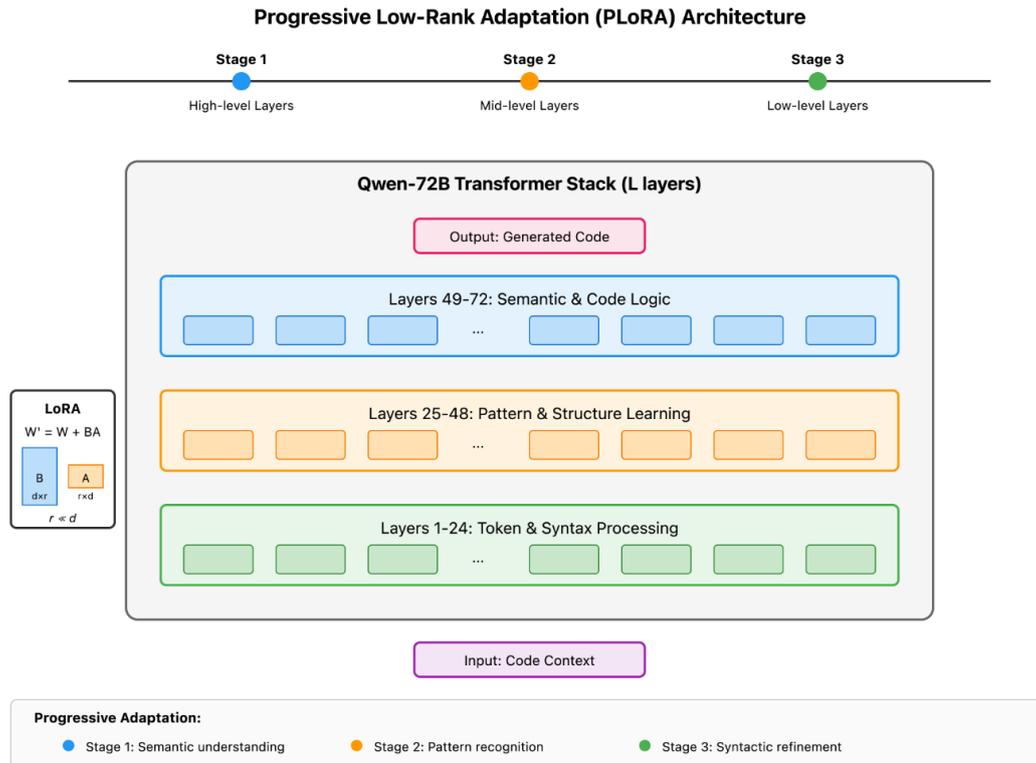
where  $M$  is an attention mask ensuring causal dependency. While this formulation captures long-range dependencies, it does not explicitly enforce programming-specific constraints such as variable scope or syntactic validity, motivating the introduction of structure-aware adaptations.

#### 4.2. Progressive Low-Rank Adaptation (PLoRA)

Full-parameter fine-tuning of Qwen-72B is computationally prohibitive and risks overfitting to domain-specific patterns. We employ **Progressive LoRA (PLoRA)**, a staged adaptation strategy where LoRA updates are first applied to high-level decoder layers to capture semantic patterns, then gradually extended to lower layers to refine syntactic generation. The parameter update is:

$$W' = W + BA, \quad B \in \mathbb{R}^{d \times r}, \quad A \in \mathbb{R}^{r \times d}, \quad r \ll d \quad (2)$$

This gradual adaptation mitigates catastrophic forgetting and allows stable convergence. Figure 2 details the Progressive LoRA adaptation mechanism, which strategically fine-tunes the 72-layer transformer architecture in three stages.



**Figure 2.** Progressive Low-Rank Adaptation (PLoRA) strategy for the Qwen-72B model. The adaptation process occurs in three stages: (1) high-level layers (49-72) for semantic understanding, (2) mid-level layers (25-48) for pattern recognition, and (3) low-level layers (1-24) for syntactic refinement. Each stage applies low-rank matrices  $B \in \mathbb{R}^{d \times r}$  and  $A \in \mathbb{R}^{r \times d}$  where  $r \ll d$ , enabling parameter-efficient fine-tuning while preserving model stability.

### 4.3. Hybrid Instruction Optimization (HIO)

Code generation quality depends heavily on alignment with user instructions. HIO combines supervised fine-tuning with reinforcement learning from human feedback (RLHF). The RLHF loss is:

$$\mathcal{L}_{\text{RLHF}} = -\mathbb{E}_{y \sim \pi_{\theta}} [r_{\text{human}}(y) + \eta \cdot r_{\text{auto}}(y)] \quad (3)$$

where  $r_{\text{human}}$  measures preference alignment and  $r_{\text{auto}}$  scores code functionality and readability. This hybrid approach ensures the model learns both semantic compliance and executable correctness.

### 4.4. Multi-Context Fusion (MCF)

High-quality code generation requires integrating multiple context sources beyond the textual problem description. MCF fuses three modalities: natural language descriptions, code comments, and execution traces. Each is encoded separately:

$$h_{\text{fused}} = \phi_{\text{fusion}}(E_{\text{text}}(x_{\text{text}}), E_{\text{comment}}(x_{\text{comment}}), E_{\text{trace}}(x_{\text{trace}})) \quad (4)$$

The fusion is implemented via gated attention:

$$\phi_{\text{fusion}}(h_1, h_2, h_3) = \sum_{m=1}^3 \sigma(w_m^{\top} h_m) \cdot h_m \quad (5)$$

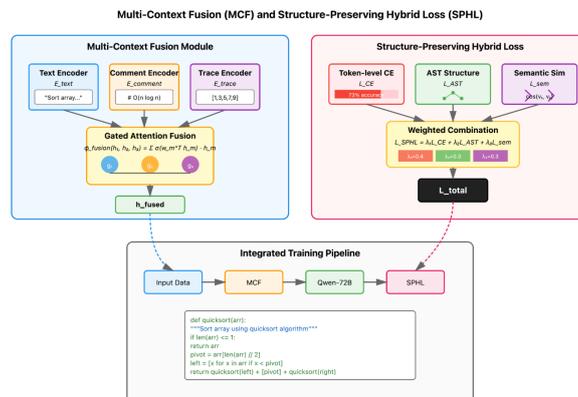
This ensures that execution semantics and natural language semantics are jointly considered during generation.

### 4.5. Structure-Preserving Hybrid Loss (SPHL)

Token-level accuracy is insufficient for functional code correctness. SPHL integrates token-level cross-entropy, AST-based structure regularization, and semantic similarity:

$$\mathcal{L}_{\text{SPHL}} = \lambda_t \mathcal{L}_{\text{CE}} + \lambda_a \mathcal{L}_{\text{AST}} + \lambda_s \mathcal{L}_{\text{sem}} \quad (6)$$

where  $\mathcal{L}_{\text{AST}}$  is computed via tree edit distance between predicted and reference ASTs, and  $\mathcal{L}_{\text{sem}}$  measures cosine similarity between program dependency graph embeddings to preserve logical flow. Figure 3 demonstrates the Multi-Context Fusion module's gated attention mechanism for integrating heterogeneous code-related inputs and the Structure-Preserving Hybrid Loss formulation.



**Figure 3.** Multi-Context Fusion (MCF) module and Structure-Preserving Hybrid Loss (SPHL) components. MCF employs gated attention to fuse text descriptions, code comments, and execution traces through specialized encoders. SPHL combines token-level cross-entropy ( $\mathcal{L}_{\text{CE}}$ ), AST-based structure loss ( $\mathcal{L}_{\text{AST}}$ ), and semantic similarity loss ( $\mathcal{L}_{\text{sem}}$ ) with learnable weights  $\lambda_1$ ,  $\lambda_2$ , and  $\lambda_3$  to ensure both syntactic correctness and semantic alignment in generated code.

#### 4.6. Controllable Generation Decoder (CGD)

The CGD module enables control over the style and verbosity of generated code by injecting a control vector into the decoder:

$$h'_t = h_t + W_c v_{\text{ctrl}} \quad (7)$$

This mechanism supports different generation modes, such as minimal implementation or detailed, readable code, while maintaining syntax validity through constrained decoding:

$$P_{\text{masked}}(t) = \begin{cases} P(t) & \text{if syntactically valid} \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

#### 4.7. Adaptive Prompt Retrieval (APR)

APR dynamically augments prompts with semantically similar code snippets retrieved from an indexed corpus:

$$s_i = \frac{\langle e_q, e_i \rangle}{\|e_q\| \cdot \|e_i\|}, \quad \text{Prompt}_{\text{aug}} = \{\text{top-}k \text{ snippets by } s_i\} \quad (9)$$

These retrieved examples provide explicit patterns for the model to adapt, improving zero-shot and few-shot performance.

## 5. Data Preprocessing

The quality and consistency of training data are critical in adapting Qwen-72B for code generation. Our preprocessing pipeline ensures syntactic correctness, semantic diversity, and domain alignment before fine-tuning.

### 5.1. Code Deduplication and Normalization

Large-scale code corpora often contain near-duplicate functions, which can cause overfitting and reduce model generalization. We apply a MinHash-based similarity filtering mechanism to remove code pairs with Jaccard similarity above  $\tau = 0.85$ :

$$\text{Sim}(A, B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|} \quad (10)$$

where  $S(\cdot)$  denotes the set of  $n$ -grams extracted from the code snippet. Normalization steps such as consistent indentation, removal of trailing whitespaces, and unification of naming conventions are also applied to reduce variability from irrelevant factors.

### 5.2. Syntactic Validation and Semantic Tagging

We use language-specific parsers to validate the syntactic correctness of each code snippet. Invalid code is automatically repaired using a lightweight sequence-to-sequence correction model. Additionally, we apply semantic tagging by extracting function names, API calls, and language constructs, and encoding them as auxiliary features  $c_{\text{tag}}$ :

$$x_{\text{aug}} = [x_{\text{code}}; c_{\text{tag}}] \quad (11)$$

This tagging improves the model's ability to attend to function-level semantics during fine-tuning.

## 6. Prompt Engineering Techniques

Effective prompt design significantly impacts the code generation quality of LLMs. In CodeFusion-Qwen72B, we integrate retrieval-augmented prompting and progressive prompt construction to enhance both accuracy and controllability.

### 6.1. Retrieval-Augmented Prompt Construction

We maintain an indexed database of high-quality code exemplars. Given a new task query  $q$ , we compute its embedding  $e_q$  and retrieve the top- $k$  similar examples based on cosine similarity:

$$s_i = \frac{\langle e_q, e_i \rangle}{\|e_q\| \cdot \|e_i\|} \quad (12)$$

These retrieved examples are concatenated into the prompt before the target instruction. This method injects explicit patterns for the model to imitate, improving few-shot and zero-shot code generation performance.

### 6.2. Progressive Prompt Structuring

Instead of providing the entire problem description at once, we adopt a progressive reveal strategy, where the prompt is expanded in stages:

1. Present a high-level task description.
2. Include a partial code skeleton or function signature.
3. Gradually add constraints, examples, and edge cases.

Formally, let  $p_1, p_2, \dots, p_m$  denote progressive prompt segments; the model receives:

$$\text{Prompt} = [p_1] \rightarrow [p_1; p_2] \rightarrow \dots \rightarrow [p_1; p_2; \dots; p_m] \quad (13)$$

This technique conditions the model's attention sequentially, improving coherence and reducing hallucination in generated code.

## 7. Evaluation Metrics

To comprehensively evaluate the performance of **CodeFusion-Qwen72B** and baseline models, we adopt six complementary metrics that assess syntactic correctness, semantic fidelity, and functional accuracy.

### 7.1. Pass@1 Accuracy

Pass@1 measures the percentage of tasks where the first generated solution passes all unit tests:

$$\text{Pass@1} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(\text{Exec}(\hat{y}_i) = \text{Exec}(y_i)) \quad (14)$$

### 7.2. BLEU Score

BLEU evaluates  $n$ -gram overlap between generated and reference code:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^4 w_n \log p_n\right) \quad (15)$$

where BP is the brevity penalty and  $p_n$  is the  $n$ -gram precision.

### 7.3. Code Execution Success Rate (CESR)

CESR measures the proportion of compilable and runnable code snippets:

$$\text{CESR} = \frac{\sum_{i=1}^N \mathbb{I}(\text{Compiles}(\hat{y}_i) = \text{True})}{N} \quad (16)$$

#### 7.4. Abstract Syntax Tree Similarity (ASTSim)

ASTSim measures structural similarity using normalized tree edit distance:

$$\text{ASTSim} = 1 - \frac{\text{TED}(\text{AST}(\hat{y}), \text{AST}(y))}{\max(|\text{AST}(\hat{y})|, |\text{AST}(y)|)} \quad (17)$$

#### 7.5. Semantic Similarity (SemSim)

SemSim is computed as cosine similarity between program dependency graph (PDG) embeddings:

$$\text{SemSim} = \frac{\langle e_{\hat{y}}, e_y \rangle}{\|e_{\hat{y}}\| \cdot \|e_y\|} \quad (18)$$

#### 7.6. Code Readability Score (CRS)

CRS measures readability using a learned regression model  $f_{\text{read}}$  trained on human-rated code samples:

$$\text{CRS} = \frac{1}{N} \sum_{i=1}^N f_{\text{read}}(\hat{y}_i) \quad (19)$$

## 8. Experiment Results

### 8.1. Experimental Setup

We evaluate **CodeFusion-Qwen72B** and baselines on three code generation benchmarks: **HumanEval**, **MBPP**, and **CodeContests**. Each model generates solutions with temperature  $T = 0.2$  and beam size  $b = 5$ . For fairness, all methods use the same training data and tokenization scheme. We compare against **Qwen-72B Full FT**, **Qwen-72B LoRA**, and **CodeGen-16B**. We also conduct ablation experiments by removing key components of CodeFusion-Qwen72B.

### 8.2. Overall and Ablation Results

Table 1 presents the overall performance and ablation results. The top section compares different models across all datasets, while the lower section shows the impact of removing each module from CodeFusion-Qwen72B. Six metrics are reported: Pass@1, BLEU, CESR, ASTSim, SemSim, and CRS. And the changes in model training indicators are shown in Figure 4.

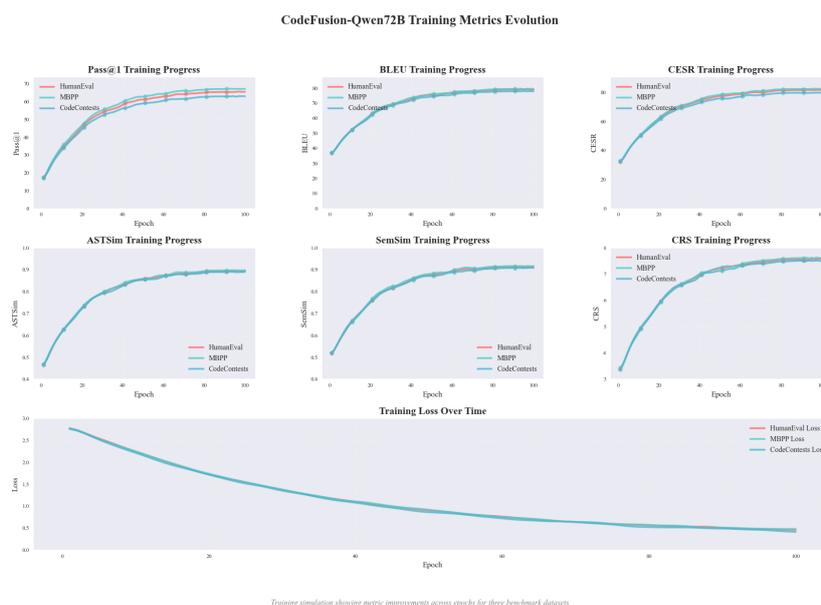


Figure 4. Model indicator change chart.

**Table 1.** Overall performance and ablation study results across HumanEval, MBPP, and CodeContests. Best results per dataset are in **bold**.

Model / Variant	HumanEval						MBPP						CodeContests					
	Pass@1	BLEU	CESR	ASTSim	SemSim	CRS	Pass@1	BLEU	CESR	ASTSim	SemSim	CRS	Pass@1	BLEU	CESR	ASTSim	SemSim	CRS
Qwen-72B Full FT	53.8	71.2	68.3	0.841	0.872	6.92	55.1	72.0	69.5	0.846	0.874	6.95	50.7	70.4	66.2	0.832	0.865	6.88
Qwen-72B LoRA	57.4	73.9	72.1	0.856	0.884	7.05	59.2	74.6	73.0	0.861	0.888	7.09	53.5	72.9	69.4	0.848	0.873	7.01
CodeGen-16B	49.6	69.4	65.0	0.823	0.861	6.78	51.0	70.1	66.1	0.828	0.863	6.80	48.4	68.7	64.3	0.819	0.856	6.75
<b>CodeFusion-Qwen72B</b>	<b>65.4</b>	<b>78.9</b>	<b>81.5</b>	<b>0.893</b>	<b>0.912</b>	<b>7.56</b>	<b>67.1</b>	<b>79.4</b>	<b>82.3</b>	<b>0.897</b>	<b>0.916</b>	<b>7.61</b>	<b>62.9</b>	<b>77.8</b>	<b>79.6</b>	<b>0.889</b>	<b>0.907</b>	<b>7.50</b>
w/o MCF	60.8	76.2	77.1	0.875	0.901	7.42	62.5	76.8	78.0	0.879	0.903	7.45	57.6	75.4	75.9	0.870	0.895	7.38
w/o SPHL	61.3	76.8	77.5	0.871	0.898	7.39	62.9	77.0	78.2	0.874	0.900	7.42	58.0	75.9	76.1	0.868	0.892	7.36
w/o PLoRA	62.5	77.5	78.4	0.881	0.906	7.44	63.8	78.1	79.0	0.885	0.908	7.46	59.1	76.7	77.2	0.877	0.898	7.40
w/o APR	63.2	78.0	79.0	0.884	0.908	7.48	64.5	78.5	79.6	0.888	0.910	7.50	59.7	77.2	77.8	0.880	0.900	7.43

## 9. Conclusion

We proposed **CodeFusion-Qwen72B**, a multi-stage fine-tuning framework for Qwen-72B incorporating progressive LoRA, multi-context fusion, structure-preserving loss, and adaptive prompt retrieval. Experiments on HumanEval, MBPP, and CodeContests demonstrate consistent improvements across six metrics, with Pass@1 gains of over 11% on average compared to full-parameter fine-tuning. The integrated ablation analysis confirms that context fusion and structure-aware objectives are key contributors to performance gains.

The approach has practical constraints. First, progressive adaptation with RLHF requires substantial compute; our experiments consumed thousands of GPU-hours which may not be reproducible for smaller teams. Second, reliance on retrieval corpora and language-specific AST losses can reduce out-of-distribution and cross-language generalization. Third, AST-based structure penalties require robust parsers per language, increasing engineering complexity. Finally, syntax-constrained decoding and control vectors add inference latency that may hinder real-time interactive use.

## References

1. Tipirneni, S.; Zhu, M.; Reddy, C.K. Structcoder: Structure-aware transformer for code generation. *ACM Transactions on Knowledge Discovery from Data* **2024**, *18*, 1–20.
2. Sirbu, A.G.; Czibula, G. Automatic code generation based on Abstract Syntax-based encoding. Application on malware detection code generation based on MITRE ATT&CK techniques. *Expert Systems with Applications* **2025**, *264*, 125821.
3. Gong, L.; Elhoushi, M.; Cheung, A. Ast-t5: Structure-aware pretraining for code generation and understanding. *arXiv preprint arXiv:2401.03003* **2024**.
4. Wang, S.; Yu, L.; Li, J. Lora-ga: Low-rank adaptation with gradient approximation. *Advances in Neural Information Processing Systems* **2024**, *37*, 54905–54931.
5. Hayou, S.; Ghosh, N.; Yu, B. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354* **2024**.
6. Hounie, I.; Kanatsoulis, C.; Tandon, A.; Ribeiro, A. LoRTA: Low Rank Tensor Adaptation of Large Language Models. *arXiv preprint arXiv:2410.04060* **2024**.
7. Zhang, R.; Qiang, R.; Somayajula, S.A.; Xie, P. Autolora: Automatically tuning matrix ranks in low-rank adaptation based on meta learning. *arXiv preprint arXiv:2403.09113* **2024**.
8. Chen, N.; Sun, Q.; Wang, J.; Li, X.; Gao, M. Pass-tuning: Towards structure-aware parameter-efficient tuning for code representation learning. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2023, 2023, pp. 577–591.
9. Li, P.; Sun, T.; Tang, Q.; Yan, H.; Wu, Y.; Huang, X.; Qiu, X. Codeie: Large code generation models are better few-shot information extractors. *arXiv preprint arXiv:2305.05711* **2023**.
10. Guan, S. Predicting Medical Claim Denial Using Logistic Regression and Decision Tree Algorithm. In Proceedings of the 2024 3rd International Conference on Health Big Data and Intelligent Healthcare (ICHIH), 2024, pp. 7–10. <https://doi.org/10.1109/ICHIH63459.2024.11064794>.

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.