Article

Not peer-reviewed version

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

Rui Guo *

Posted Date: 21 October 2025

doi: 10.20944/preprints202509.2169.v2

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

 doi:10.20944/preprints202509.2169.v2

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. Hounie et al. [5] extended this with LoRTA, which

 doi:10.20944/preprints202509.2169.v2

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

Li et al. [8] showed that large code models work well as few-shot information extractors.

3. Methodology

Better code generation in ultra-large language models depends on parameter-efficient tuning and structure-aware 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. HIO combines supervised fine-tuning with RLHF to align functionality and readability. MCF augments inputs with problem descriptions, comments, and execution traces. SPHL enforces token, syntactic, and semantic coherence. CGD and syntax-constrained decoding produce controllable, compilable outputs. APR retrieves relevant code snippets at inference.

We evaluate on HumanEval, MBPP, and CodeContests. Versus a fully fine-tuned Qwen-72B baseline, pass@1 improves from 53.8% to 65.4% (+11.6), BLEU from 71.2 to 78.9 (+7.7), and execution success from 68.3% to 81.5% (+13.2). Ablation studies indicate MCF and SPHL drive the largest execution gains, while PLoRA and HIO yield substantial syntactic and semantic improvements. The pipeline is shown in Figure 1

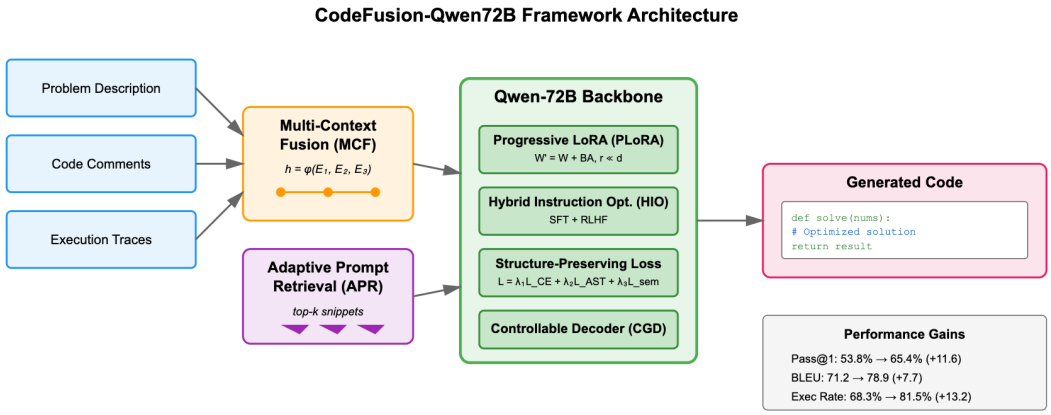

Figure 1. Overview of the CodeFusion-Qwen72B framework architecture. The system integrates Multi-Context Fusion (MCF) for processing diverse input modalities, Adaptive Prompt Retrieval (APR) for dynamic context augmentation, and a multi-component Qwen-72B backbone enhanced with Progressive LoRA (PLoRA), 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 **CodeFusion-Qwen72B** framework is a multi-stage, multi-objective fine-tuning pipeline built on the Qwen-72B autoregressive decoder. This section concisely describes its components, motivations, and implementation essentials.

4.1. Qwen-72B Backbone Architecture

Qwen-72B employs Ldecoder-only transformer layers with multi-head self-attention and feed-forward blocks. The attention is computed as

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

 doi:10.20944/preprints202509.2169.v2

where M enforces causality. This backbone captures long-range dependencies but lacks explicit programming constraints such as scope and syntactic validity, motivating structure-aware adaptations.

4.2. Progressive Low-Rank Adaptation (PLoRA)

Full fine-tuning is costly and prone to overfitting. We apply Progressive LoRA (PLoRA), which stages low-rank updates from higher to lower layers to capture semantics first, then refine syntax:

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

This staged schedule reduces catastrophic forgetting and improves stability. Figure 2 diagrams the three adaptation stages.

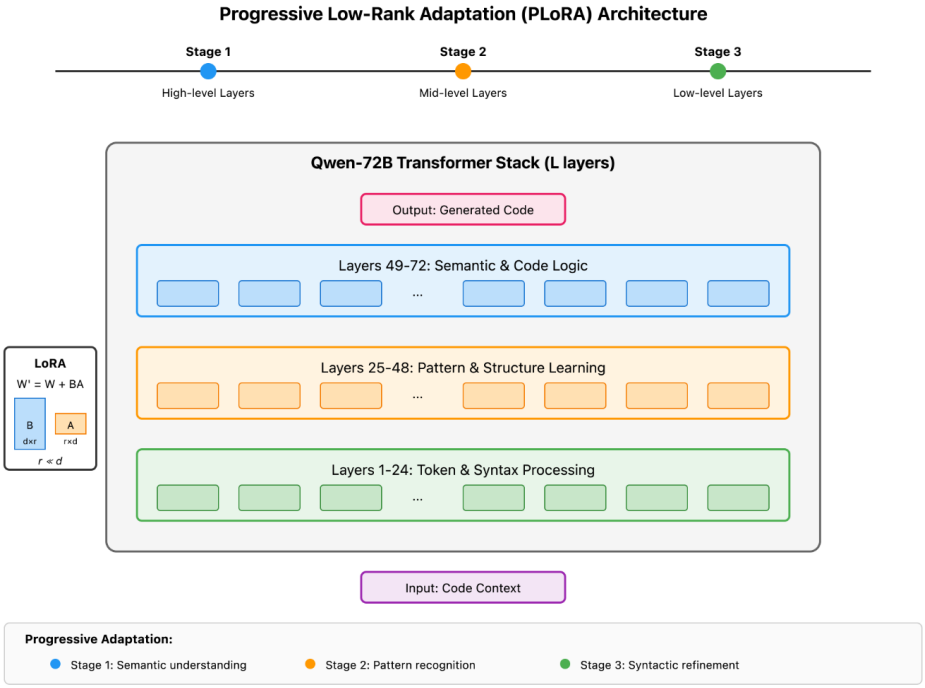

Figure 2. Progressive Low-Rank Adaptation (PLoRA) strategy for the Qwen-72B model. The adaptation process occurs in three stages: (1) high-level layers (49-72) for semantic understanding, (2) mid-level layers (25-48) for pattern recognition, and (3) low-level layers (1-24) for syntactic refinement. Each 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)

HIO aligns model outputs with human preferences and code functionality by combining supervised fine-tuning and RLHF. The RLHF objective is

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

where r_{human} captures preference alignment and r_{auto} scores automated functionality and readability.

4.4. Multi-Context Fusion (MCF)

MCF fuses problem text, comments, and execution traces to provide richer context. Encoders produce modality embeddings which are combined via gated attention:

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

with

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

 doi:10.20944/preprints202509.2169.v2

This joint representation injects execution semantics alongside natural language into decoding.

4.5. Structure-Preserving Hybrid Loss (SPHL)

SPHL combines token, structural, and semantic objectives to prioritize functional correctness:

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

Here $\mathcal{L}_{\text{AST}}$ is derived from tree edit distance between predicted and reference ASTs and $\mathcal{L}_{\text{sem}}$ measures cosine similarity between program dependency graph embeddings. Figure 3 summarizes MCF and SPHL components.

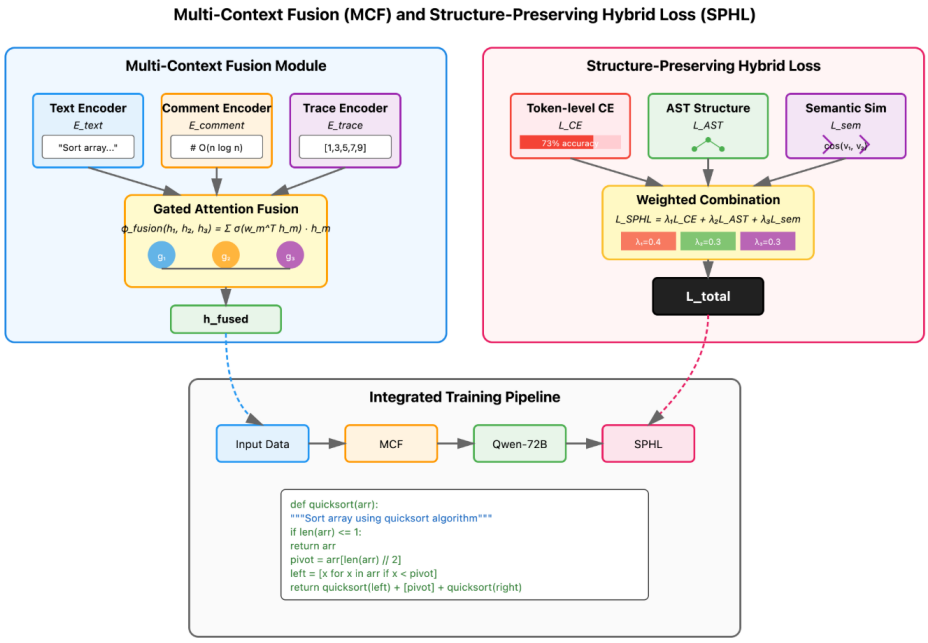

Figure 3. Multi-Context Fusion (MCF) module and Structure-Preserving Hybrid Loss (SPHL) components. MCF employs gated attention to fuse text descriptions, code comments, and execution traces through specialized encoders. SPHL combines token-level cross-entropy ($\mathcal{L}_{\text{CE}}$), AST-based structure loss ($\mathcal{L}_{\text{AST}}$), and semantic similarity loss ($\mathcal{L}_{\text{sem}}$) with learnable weights λ_1, λ_2, and λ_3 to ensure both syntactic correctness and semantic alignment in generated code.

4.6. Controllable Generation Decoder (CGD)

CGD injects a control vector for style and verbosity:

$$h'_t = h_t + W_c v_{\text{ctrl}} \tag{7}$$

Decoding enforces syntax via masking:

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

This permits modes from minimal implementations to detailed, readable solutions while constraining invalid tokens.

 doi:10.20944/preprints202509.2169.v2

4.7. Adaptive Prompt Retrieval (APR)

APR augments prompts with semantically similar code snippets retrieved from an indexed corpus. Similarity is measured by cosine score:

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

These examples improve zero-shot and few-shot adaptation by supplying explicit patterns.
Overall, CodeFusion-Qwen72B integrates PLoRA, HIO, MCF, SPHL, CGD, and APR to balance parameter efficiency, instruction alignment, multi-context awareness, structural fidelity, controllability, and dynamic retrieval.

5. Data Preprocessing

Data quality and consistency are enforced by a preprocessing pipeline that prioritizes syntactic validity, semantic diversity, and domain alignment prior to fine-tuning.

5.1. Code Deduplication and Normalization

Near-duplicate functions are removed via MinHash-based similarity filtering with threshold $\tau = 0.85$. Similarity is measured as

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

where $S(\cdot)$ denotes n-gram sets. Standard normalization is applied, including consistent indentation, trimming trailing whitespace, and unifying naming conventions to reduce spurious variability.

5.2. Syntactic Validation and Semantic Tagging

Language-specific parsers validate syntax. Invalid snippets are auto-repaired using a lightweight sequence-to-sequence correction model. We extract semantic tags (function names, API calls, language constructs) and append them as auxiliary features:

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

This augmentation improves function-level attention and downstream fine-tuning.

6. Prompt Engineering Techniques

Prompt design affects LLM code generation. CodeFusion-Qwen72B uses retrieval-augmented prompting and progressive prompt construction to improve accuracy and controllability.

6.1. Retrieval-Augmented Prompt Construction

An indexed corpus of high-quality exemplars is queried for a task embedding e_q. We rank snippets by cosine similarity and prepend the top-k examples to the instruction:

$$s_i = \frac{\langle e_q, e_i \rangle}{\|e_q\| \cdot \|e_i\|} \tag{12}$$

These exemplars supply explicit patterns that boost few-shot and zero-shot performance.

6.2. Progressive Prompt Structuring

Prompts are revealed in stages to guide attention and reduce hallucination. Let $p_1, \ldots, p_m$ be successive segments. The model receives:

$$\text{Prompt} = [p_1] \to [p_1; p_2] \to \cdots \to [p_1; \ldots; p_m] \tag{13}$$

 doi:10.20944/preprints202509.2169.v2

Typical progression: (1) high-level description, (2) function signature or skeleton, (3) constraints, examples, and edge cases. This sequential conditioning improves coherence and alignment with task requirements.

7. Evaluation Metrics

We evaluate CodeFusion-Qwen72B and baselines using six complementary metrics targeting syntactic validity, structural fidelity, semantic alignment, and functional correctness.

7.1. Pass@1 Accuracy

Pass@1 is the fraction of tasks whose first generated solution passes all unit tests:

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

7.2. BLEU Score

BLEU measures n-gram overlap between candidate and reference code with brevity penalty:

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

7.3. Code Execution Success Rate (CESR)

CESR reports the proportion of generated snippets that compile and run:

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

7.4. Abstract Syntax Tree Similarity (ASTSim)

ASTSim quantifies structural similarity via normalized tree edit distance:

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

7.5. Semantic Similarity (SemSim)

SemSim is the cosine similarity between program dependency graph embeddings:

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

7.6. Code Readability Score (CRS)

CRS is the mean output of a learned readability regressor trained on human ratings:

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

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.

 doi:10.20944/preprints202509.2169.v2

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.

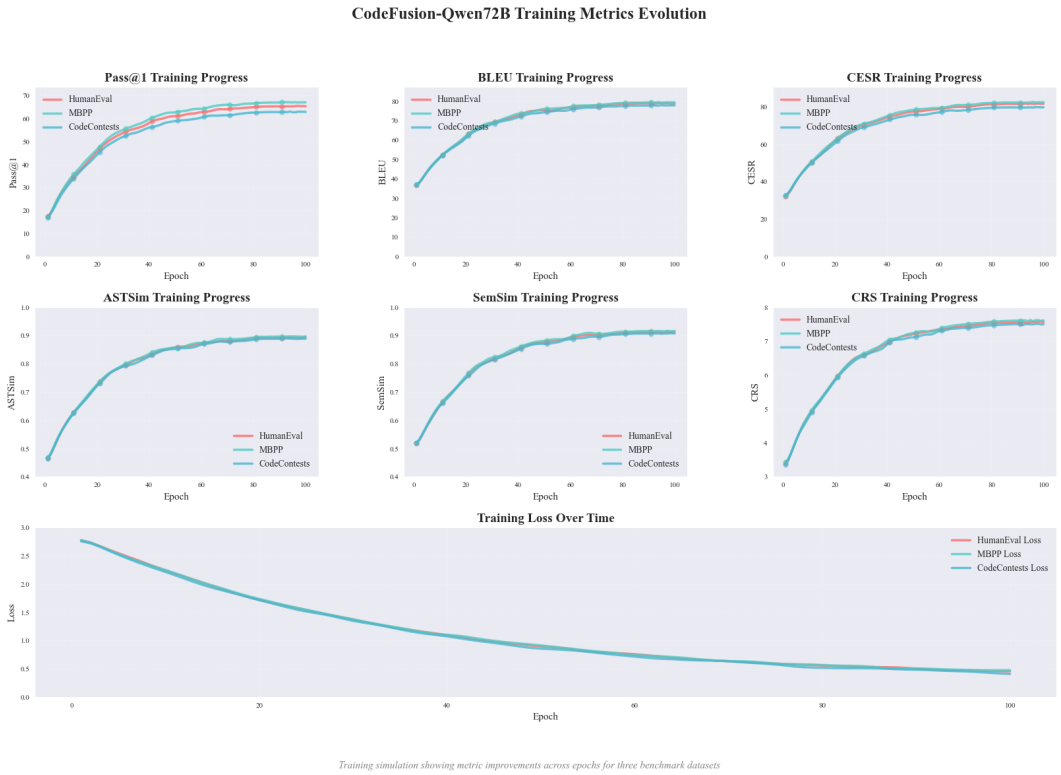

Figure 4. Model indicator change chart.

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
CodeFusion-Qwen72B	**65.4**	**78.9**	**81.5**	**0.893**	**0.912**	**7.56**	**67.1**	**79.4**	**82.3**	**0.897**	**0.916**	**7.61**	**62.9**	**77.8**	**79.6**	**0.889**	**0.907**	**7.50**
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. Conclusions

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.

 doi:10.20944/preprints202509.2169.v2

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. Hounie, I.; Kanatsoulis, C.; Tandon, A.; Ribeiro, A. LoRTA: Low Rank Tensor Adaptation of Large Language Models. *arXiv preprint arXiv:2410.04060* **2024**.
6. 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**.
7. 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.
8. 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**.