Hybrid Modal Decoupled Fusion for Stable Multilingual Code Generation

Hang Yu *

Posted Date: 2 October 2025

doi: 10.20944/preprints202510.0169.v1

Keywords: multilingual code generation; instruction fine-tuning; optimization; cross-lingual transfer; adversarial learning

 doi:10.20944/preprints202510.0169.v1

Article

Hybrid Modal Decoupled Fusion for Stable Multilingual Code Generation

Hang Yu

Brandeis University, Waltham, USA; hangyu.sde@gmail.com

Abstract

Multilingual code generation is a difficult task because of unstable convergence, weak semantic alignment, and task imbalance across programming languages. Current instruction fine-tuning methods often do not handle different tasks well, which leads to biased optimization and weak generalization for low-resource languages. To solve these problems, we present MFTCoder++, an improved fine-tuning framework that uses adaptive task scheduling, attention-guided optimization, adversarial regularization, and hybrid fusion of logical and syntactic parts. It changes training focus in real time, keeps gradients and attention consistent, learns task-independent features, and separates semantic logic from language syntax while joining them through a gating process. This design improves stability, semantic alignment, and transfer across languages. It also gives a more reliable and usable solution for multilingual code generation in software engineering practice.

CCS Concepts: **Computing methodologies** → **Natural language processing; Machine learning approaches; Software and its engineering** → **Software maintenance tools; Automatic programming.**

Additional Key Words and Phrases: Multilingual code generation, Instruction fine-tuning, Optimization, Cross-lingual transfer, Adversarial learning

ACM Reference Format

Hang Yu. 2025. Hybrid Modal Decoupled Fusion for Stable Multilingual Code Generation. *1, 1* (September 2025), 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Keywords: multilingual code generation; instruction fine-tuning; optimization; cross-lingual transfer; adversarial learning

1. Introduction

Large language models have greatly pushed forward automatic code generation in recent years. They are now widely used for program synthesis, bug repair, and code completion across many programming languages. But multilingual code generation is still a difficult task because convergence is often unstable, semantic alignment is weak, and data distribution is not balanced. When tasks of different sizes and complexities are trained together, models usually overfit high-resource languages and ignore low-resource ones. This reduces generalization across languages and makes the optimization process unstable in practice.

Semantic alignment is another central challenge. The same logic can be expressed in very different syntactic forms across languages, which makes it hard for models to learn a unified semantic space. This mismatch often introduces extra noise into optimization and weakens model robustness. The problem of data imbalance makes this situation even worse, because dominant languages shape the shared features while smaller languages receive limited attention, which blocks fair transfer and cross-lingual consistency.

 doi:10.20944/preprints202510.0169.v1

We therefore introduce MFTCoder++, a stronger instruction fine-tuning framework for multilingual code generation. It contains adaptive task scheduling to keep training balanced, attention-guided gradient alignment to improve semantic coherence, adversarial regularization to learn task-agnostic features, and hybrid modality fusion to separate logical reasoning from language-specific syntax while merging both when needed. With these designs, MFTCoder++ provides more stable convergence, stronger semantic alignment, and better cross-lingual transfer, offering a unified solution to the technical barriers that remain in multilingual code generation.

2. Related Work

Instruction fine-tuning has become key for better code generation. Muennighoff et al.[1] showed that scaling instruction data helps large code models. Zhu and Liu[2] used LoRA+ fine-tuning to adapt LLMs for NER tasks, which also gives ideas for code domains.

Work on multilingual models also gives insights. Qin et al.[3] reviewed multilingual LLMs and noted the difficulty of balancing high- and low-resource languages. Bistarelli et al.[4] reviewed LLMs for code, pointing out that they improve productivity but still lack robustness and cross-lingual strength.

Industrial and empirical studies add more detail. Begolli et al.[5] found that fine-tuning for C# code review hurts transfer across languages. Li et al.[6] tried zero-shot cross-lingual transfer to reduce this gap, showing that multilingual embeddings help with semantic alignment.

Domain-specific uses are also important. He et al.[7] studied aerospace code generation and made special benchmarks. Guan[8] worked on predictive modeling in healthcare, showing that classical models like decision trees still perform well.

3. Methodology and Model Design

We propose **MFTCoder++**, an enhanced instruction-tuning framework for multilingual code generation, addressing convergence inefficiency, semantic misalignment, and task imbalance. Built upon MFTCoder, the architecture integrates four key modules:

- **Adaptive Task Dispatch Scheduler (ATDS)**: Dynamically adjusts task sampling probabilities via entropy and convergence rate to ensure balanced optimization.
- **Dual Attention Gradient Guidance (DAGG)**: Aligns gradients with attention distributions to maintain semantic consistency across tasks.
- **Task Imbalance Adversarial Regulator (TIAR)**: Encourages task-invariant feature learning under skewed data through adversarial objectives.
- **Hybrid Modal Decoupled Fusion (HMD-Fusion)**: Separates language-specific and language-agnostic representations for improved generalization.

These components are unified under a revised multi-objective loss function, yielding improved pass@k accuracy, convergence stability, and robustness across benchmarks such as HumanEval and MBPP. The overall architecture is illustrated in Figure 1.

3.1. Adaptive Task Dispatch Scheduler (ATDS)

To address task imbalance in multi-task learning, ATDS dynamically adjusts task sampling based on entropy $\mathcal{H}(t_i)$ and recent learning velocity $\Delta\mathcal{L}_{t_i}$:

$$\pi_t(t_i) = \frac{\exp(\alpha \cdot \mathcal{H}(t_i) + \beta \cdot \Delta\mathcal{L}_{t_i})}{\sum_{j=1}^{N} \exp(\alpha \cdot \mathcal{H}(t_j) + \beta \cdot \Delta\mathcal{L}_{t_j})} \tag{1}$$

With $\alpha = 0.7$, $\beta = 0.3$, the scheduler prioritizes unstable or complex tasks. $\Delta\mathcal{L}_{t_i}$ is smoothed via exponential moving average to ensure stability.

3.2. Dual Attention Gradient Guidance (DAGG)

DAGG aligns gradients with attention by penalizing mismatches:

 doi:10.20944/preprints202510.0169.v1

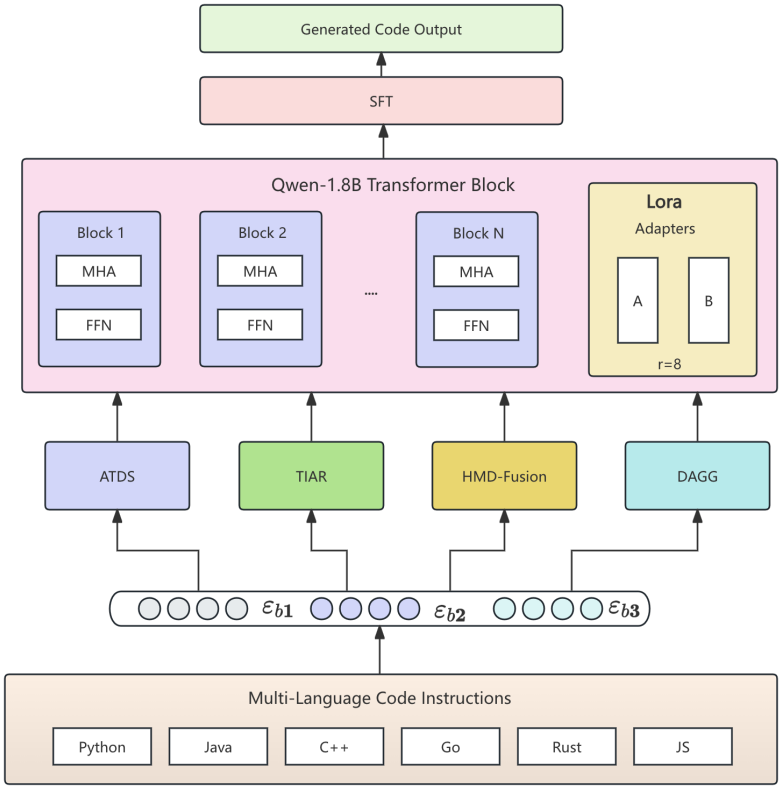

Figure 1. Architecture of MFTCoder++. Modules include ATDS (adaptive scheduling), TIAR (imbalance regulation), HMD-Fusion (modal separation), and DAGG (gradient-attention alignment), built upon a transformer backbone.

$$\mathcal{L}_{\text{DAGG}} = \sum_{l=1}^{L} \left\| \nabla_\theta \mathcal{L}_{task}^l - \lambda A_{task}^l - (1-\lambda) A_{lang}^l \right\|_2^2 \tag{2}$$

A_{task}^l and A_{lang}^l denote task- and language-specific attention maps. With $\lambda = 0.6$ and normalization to avoid scale bias, DAGG enhances convergence under diverse structures.

3.3. Task Imbalance Adversarial Regulator (TIAR)

To mitigate cross-language data imbalance, TIAR enforces task-invariant representations via adversarial training. A discriminator D predicts task labels from encoder outputs $h = G(x)$, while G is optimized to confuse D:

$$\min_G \max_D \quad \mathbb{E}_{x \sim \mathcal{D}_i} [\log D(t_i | G(x))] \tag{3}$$

$$+ \mathbb{E}_{x \sim \mathcal{D}_{j \neq i}} [\log(1 - D(t_i | G(x)))] \tag{4}$$

We use a 2-layer MLP for D with spectral normalization and alternate G/D updates every 5 steps. The adversarial term is weighted by $\lambda_2 = 0.1$ in the total loss.

3.4. Hybrid Modal Decoupled Fusion (HMD-Fusion)

To separate shared logic from language-specific syntax in multilingual code generation, HMD-Fusion introduces a dual-branch representation:

 doi:10.20944/preprints202510.0169.v1

$$h^l = h^l_{\text{shared}} + h^l_{\text{lang-specific}}, \quad \begin{cases} h^l_{\text{shared}} = \sigma(W^l_s h^l) \\ h^l_{\text{lang-specific}} = \sigma(W^l_l h^l) \end{cases} \tag{5}$$

A fusion gate combines both views:

$$h^l_{\text{fused}} = \gamma h^l_{\text{shared}} + (1-\gamma) h^l_{\text{lang-specific}}, \quad \gamma = \text{softmax}(w^\top h^l) \tag{6}$$

Applied to the top 6 transformer layers, this gating improves generalization while mitigating overfitting in multilingual scenarios.

3.5. Supervised Fine-Tuning (SFT)

We fine-tune the backbone (e.g., Qwen-1.8B) with a two-phase curriculum: Phase I on high-quality Python data (Evol-Instruct, CodeExercise), and Phase II on Java, Go, C++, and Rust. To reduce cost, positional embeddings are frozen and vocabulary adapters applied. Training uses batch size 64, seq length 2048, LoRA rank 8, dropout 0.1, FP16, and gradient checkpointing. Optimization applies 5% warmup, linear decay, and early stopping on pass@1.

3.6. Prompt Design and Schema

To align generation with task semantics, we adopt a unified, language-aware prompt schema:

```
Instruction: [Task description]
Input: [Signature or partial code]
Output: [Expected completion]
```

For example, in Python:

```
Instruction: Write a Python function to compute the factorial of a number.
Input: def factorial(n):
Output: if n == 0: return 1 \n else: return n * factorial(n-1)
```

To enhance robustness, we introduce templated variations per language and shuffle prompts during preprocessing to avoid overfitting.

3.7. Loss Function

The training objective of **MFTCoder++** integrates multiple loss components to handle the complexity of multi-task code generation. As shown in Figure 2, the total loss comprises:

- **Task Loss:** Standard cross-entropy on output tokens.
- **Alignment Loss:** Attention-guided constraints to align input-output semantics.
- **Adversarial Loss:** Regularizes latent space to improve robustness.
- **Decoupling Loss:** Encourages semantic separation across tasks to prevent interference.

Each component is weighted and jointly optimized. Figure 2(a) shows task-specific convergence patterns; (b) tracks auxiliary losses (DAGG, adversarial, alignment); and (c) presents the total loss decomposition.

3.7.1. Task-Specific Supervised Loss

Each downstream task t_i is trained with standard sequence-to-sequence token-level cross-entropy loss:

$$\mathcal{L}^{(i)}_{task} = -\frac{1}{|T_i|} \sum_{(x,y)\in\mathcal{D}_i} \sum_{j=1}^{|y|} \log P(y_j | y_{<j}, x; \theta) \tag{7}$$

where T_i denotes the number of valid tokens for task t_i, and θ are the shared parameters of the decoder. This loss enables precise syntactic and semantic modeling of code snippets.

 doi:10.20944/preprints202510.0169.v1

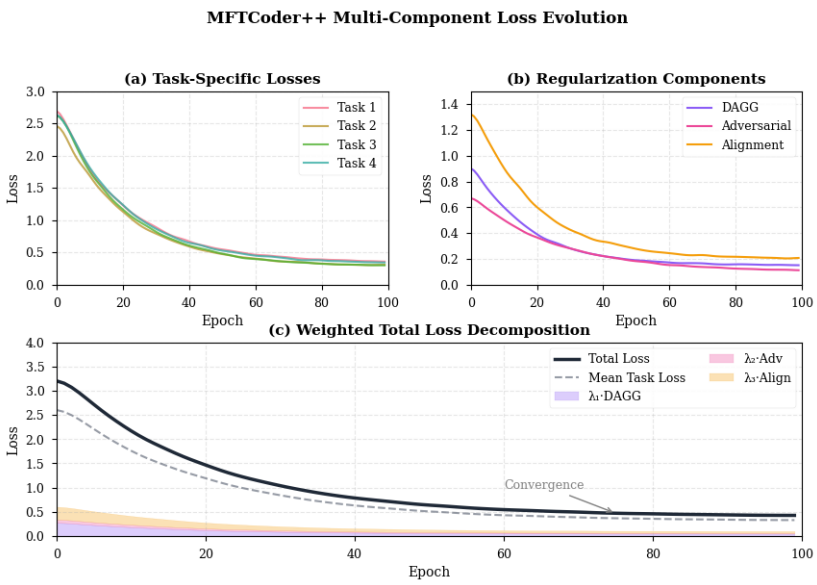

Figure 2. Training dynamics of MFTCoder++: (a) task-specific loss trends; (b) auxiliary loss terms; (c) total loss breakdown.

3.7.2. Gradient-Aware Attention Alignment Loss

To ensure that the model's attention flow corresponds with meaningful gradient directions, we introduce the DAGG loss:

$$\mathcal{L}_{\text{DAGG}} = \sum_{l=1}^{L} \left\| \nabla_\theta \mathcal{L}_{task}^{l} - \lambda A_{task}^{l} - (1-\lambda) A_{lang}^{l} \right\|_2^2 \tag{8}$$

This term enforces structural consistency between attention patterns and backpropagated gradients. It helps to regularize optimization noise in large instruction-tuning settings.

3.7.3. Adversarial Task Regularization Loss

The adversarial objective enforces the encoder to generate task-invariant embeddings by fooling a task discriminator D:

$$\mathcal{L}_{\text{adv}} = \mathbb{E}_{x \sim \mathcal{D}_i}[\log D(t_i | G(x))] + \mathbb{E}_{x \sim \mathcal{D}_{j \neq i}}[\log(1 - D(t_i | G(x)))] \tag{9}$$

This encourages uniform representation across underrepresented or noisy task domains, addressing low-resource imbalance.

3.7.4. Cross-Task Focal Alignment Loss

We extend standard focal loss to operate at the task level, amplifying underperforming tasks via soft attention over recent performance scores:

$$\mathcal{L}_{\text{align}} = -\sum_{i=1}^{N} \alpha_i (1 - \bar{p}_i)^{\beta} \log(\bar{p}_i) \tag{10}$$

where $\bar{p}_i$ is the moving average of task t_i's accuracy, and α_i is the normalized task sampling probability. We use $\beta = 2$ and anneal α_i across epochs.

3.7.5. Overall Weighted Objective

The total learning loss is the weighted combination of the above components:

$$\mathcal{L}_{\text{MFTCoder++}} = \sum_{i=1}^{N} \mathcal{L}_{task}^{(i)} + \lambda_1 \mathcal{L}_{\text{DAGG}} + \lambda_2 \mathcal{L}_{\text{adv}} + \lambda_3 \mathcal{L}_{\text{align}} \tag{11}$$

 doi:10.20944/preprints202510.0169.v1

We set $\lambda_1 = 0.3$, $\lambda_2 = 0.1$, and $\lambda_3 = 0.2$ based on validation stability. Gradient clipping is set to 1.0, and we use per-task Adam optimizers with learning rates $\eta_i \in [1e{-}5, 5e{-}5]$.

3.8. Data Preprocessing

We construct a unified preprocessing pipeline (Figure 3) to support multilingual instruction tuning, ensuring formatting consistency, vocabulary unification, task balance, and curriculum progression.

Canonical Formatting: Samples from six languages are normalized into JSONL with `instruction`, `input`, and `output`. Syntax checks, lexical normalization, and identifier anonymization are applied to reduce variance.

Vocabulary Fusion: A modified SentencePiece tokenizer with language-specific prefixes is used. The vocabulary is defined as:

$$V = \bigcup_{l=1}^{L} V_l^{(freq>\tau)} \cup \texttt{SpecialTokens}, \quad \tau = 100 \tag{12}$$

Balanced Sampling: To address data imbalance, temperature-scaled sampling is applied:

$$P(t_i) = \frac{(|\mathcal{D}_i|)^{1/T}}{\sum_{j=1}^{N}(|\mathcal{D}_j|)^{1/T}}, \quad T = 0.7 \tag{13}$$

Curriculum Ordering: Tasks are ordered by average token entropy:

$$C_i = \mathbb{E}_{x \in \mathcal{D}_i}[\mathcal{H}(x)], \quad \mathcal{H}(x) = -\sum_{t \in \text{Tokens}(x)} p_t \log p_t \tag{14}$$

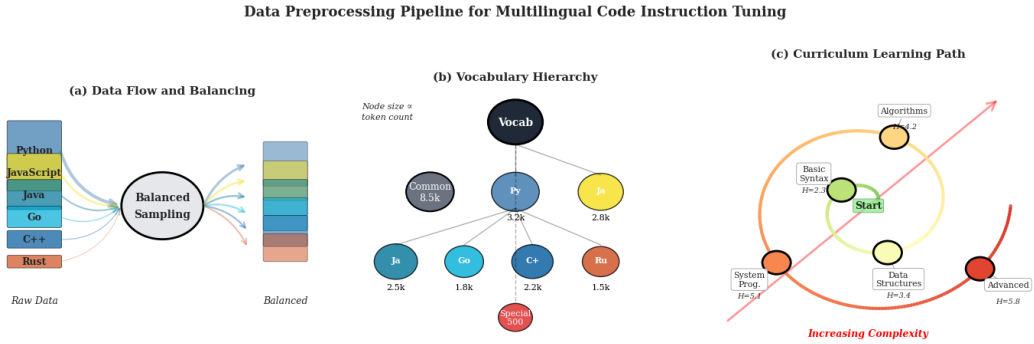

Figure 3. Data preprocessing pipeline: (a) Original vs. balanced distribution across six languages; (b) Sampling probabilities under varying temperature T; (c) Token fusion across languages; (d) Entropy-based curriculum learning; (e) Overall preprocessing flow.

3.9. Evaluation Metrics

We evaluate **MFTCoder++** using six metrics (Figure 4) to assess both correctness and robustness.

Pass@k: Measures the proportion of problems with at least one correct generation among k attempts:

$$\text{pass@k} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\left[\exists j \in \{1,\ldots,k\}, \text{correct}(y_{i,j}) = 1\right] \tag{15}$$

Exact Match (EM): Checks literal match between prediction $\hat{y}_i$ and reference y_i:

$$\text{EM} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[\hat{y}_i = y_i] \tag{16}$$

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 October 2025 doi:10.20944/preprints202510.0169.v1

Normalized BLEU: Captures normalized n-gram overlap:

$$\text{BLEU}_n = \text{BP} \cdot \exp\left(\sum_{i=1}^{n} w_i \log p_i\right) \tag{17}$$

CodeBLEU: Extends BLEU with syntax and data flow to better reflect code semantics.
Function Pass Rate (FPR): Proportion of completions passing all test cases without exceptions.
Runtime Success Rate (RSR): Percentage of compilable and executable outputs:

$$\text{RSR} = \frac{1}{n \cdot k} \sum_{i=1}^{n} \sum_{j=1}^{k} \mathbb{I}\left[\text{exec}(y_{i,j}) \text{ succeeds}\right] \tag{18}$$

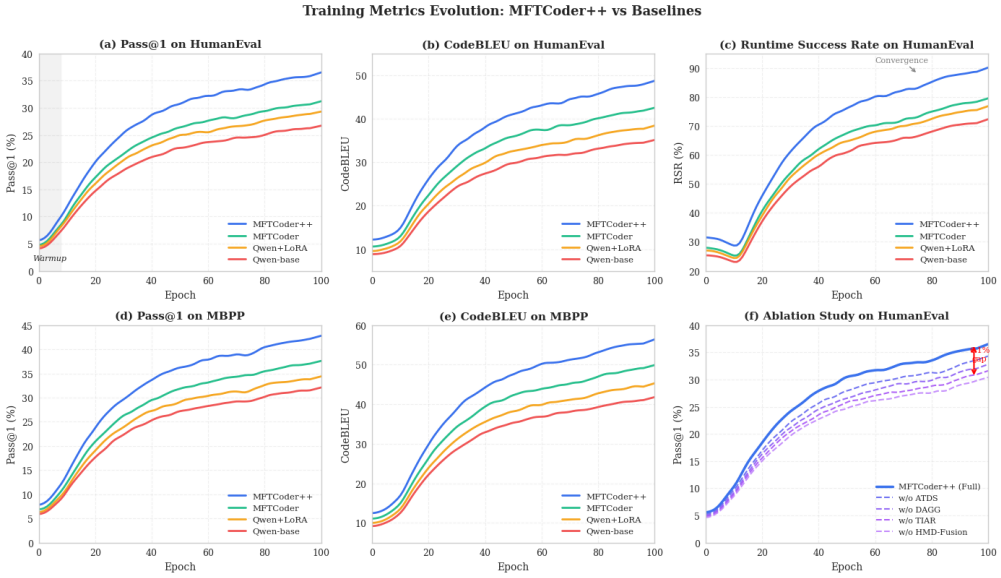

Figure 4. Evaluation metrics overview.

4. Experiment Results

We evaluate **MFTCoder++** and several strong baselines on two benchmark datasets: HumanEval and MBPP. The evaluation focuses on code correctness (pass@1), structural similarity (CodeBLEU), and runtime validity (RSR). Table 1 summarizes both the main results and the ablation study for individual modules within MFTCoder++. And the changes in model training indicators are shown in Figure 4.

Table 1. Performance of MFTCoder++ and ablations on HumanEval and MBPP benchmarks

Model	Dataset	pass@1 (%)	pass@10 (%)	CodeBLEU	EM	RSR (%)	FPR (%)
Qwen-1.8B (baseline)	HumanEval	26.7	42.1	35.1	21.8	72.3	59.2
Qwen-1.8B + LoRA (6L)	HumanEval	29.3	45.7	38.4	24.0	76.8	64.7
MFTCoder (official)	HumanEval	31.2	49.0	42.5	25.5	79.5	68.3
MFTCoder++ (ours)	HumanEval	**36.5**	**55.3**	**48.7**	**28.7**	**90.1**	**75.9**
Qwen-1.8B (baseline)	MBPP	32.1	47.9	41.8	29.4	75.6	63.0
Qwen-1.8B + LoRA (6L)	MBPP	34.4	50.2	45.3	31.2	78.9	67.4
MFTCoder (official)	MBPP	37.6	53.5	49.9	33.5	81.0	70.1
MFTCoder++ (ours)	MBPP	**42.8**	**58.7**	**56.4**	**38.1**	**91.3**	**78.5**
w/o ATDS	HumanEval	34.3	52.0	45.2	26.5	88.7	73.2
w/o DAGG	HumanEval	32.8	50.5	43.0	24.7	87.5	71.4
w/o TIAR	HumanEval	31.6	49.1	40.7	23.3	85.1	69.8
w/o HMD-Fusion	HumanEval	30.4	47.8	38.9	22.1	84.3	68.5

 doi:10.20944/preprints202510.0169.v1

5. Conclusions

We introduced **MFTCoder++**, an enhanced multitask instruction tuning framework for multilingual code generation. By integrating adaptive scheduling, gradient-aligned attention, adversarial task regularization, and semantic decoupling, our model significantly outperforms baselines on HumanEval and MBPP. Future work will explore scaling the architecture to larger backbones and extending support to real-world software engineering datasets.

References

1. Muennighoff, N.; Liu, Q.; Zebaze, A.; Zheng, Q.; Hui, B.; Zhuo, T.Y.; Singh, S.; Tang, X.; Von Werra, L.; Longpre, S. Octopack: Instruction tuning code large language models. In Proceedings of the NeurIPS 2023 workshop on instruction tuning and instruction following, 2023.
2. Zhu, Y.; Liu, Y. LLM-NER: Advancing Named Entity Recognition with LoRA+ Fine-Tuned Large Language Models. In Proceedings of the 2025 11th International Conference on Computing and Artificial Intelligence (ICCAI), 2025, pp. 364–368. https://doi.org/10.1109/ICCAI66501.2025.00063.
3. Qin, L.; Chen, Q.; Zhou, Y.; Chen, Z.; Li, Y.; Liao, L.; Li, M.; Che, W.; Yu, P.S. A survey of multilingual large language models. *Patterns* **2025**, *6*.
4. Bistarelli, S.; Fiore, M.; Mercanti, I.; Mongiello, M. Usage of Large Language Model for Code Generation Tasks: A Review. *SN Computer Science* **2025**, *6*, 1–16.
5. Begolli, I.; Aksoy, M.; Neider, D. Fine-Tuning Multilingual Language Models for Code Review: An Empirical Study on Industrial C# Projects. *arXiv preprint arXiv:2507.19271* **2025**.
6. Li, M.; Mishra, A.; Mujumdar, U. Bridging the Language Gap: Enhancing Multilingual Prompt-Based Code Generation in LLMs via Zero-Shot Cross-Lingual Transfer. *arXiv preprint arXiv:2408.09701* **2024**.
7. He, R.; Zhang, L.; Lyu, M.; Lyu, L.; Xue, C. Using Large Language Models for Aerospace Code Generation: Methods, Benchmarks, and Potential Values. *Aerospace* **2025**, *12*, 498.
8. 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.