Burmese-Coder-4B: Fine-Tuning a Small Language Model for Burmese Coding with Language-Aware Evaluation

Wai Yan Nyein Naing
waiyan.nn18@gmail.com

Abstract

Low-resource code generation remains underexplored, with existing benchmarks largely focused on English. Burmese coding presents a dual challenge: generating correct programs while maintaining natural Burmese explanations without corrupting programming syntax. In practice, models frequently exhibit mixed-script hallucination, unstable terminology, and cross-language token drift despite producing functionally correct code. We introduce **Burmese-Coder-4B**, a Burmese coding assistant adapted from the Gemma-3 4B family via supervised fine-tuning and Direct Preference Optimization (DPO). The alignment stage incorporates a language-aware penalization signal to suppress mixed-language outputs and improve linguistic consistency. Evaluation is conducted using a two-track framework combining functional correctness (Pass@1) and rubric-based **LLM-as-a-Judge** scoring using Gemini 2.5 and DeepSeek V3. Burmese-Coder-4B matches the base model on Pass@1 (62.0%) while improving rubric quality (3.779 vs. 3.203) and reducing mixed-language penalty (0.02 vs. 0.69). Compared to Qwen2.5-3B (45.0% Pass@1, 2.526 rubric), the burmese-coder model achieves stronger alignment in both code correctness and Burmese language quality. These results demonstrate that functional correctness alone is insufficient for evaluating low-resource coding assistants. We release the model and evaluation framework to support reproducible research.[1]

1 Introduction

Recent progress in code generation has been driven primarily by English-centric benchmarks, leaving low-resource languages substantially underexplored. Burmese presents a particularly challenging setting because coding assistants must operate across two distinct token regimes: natural-language explanation in Burmese and standardized programming syntax that should remain untranslated. In practice, this leads to a recurring failure mode in which a model may generate functionally correct code while still producing poor user-facing output through unstable terminology, mixed-script hallucination, or incorrect translation of code-related expressions.

These issues are especially pronounced in small language models and low-resource settings, where next-token generation may drift into nearby high-frequency scripts and languages observed during pretraining. As a result, conventional evaluation based only on unit-test correctness can overestimate real usability for native Burmese speakers. A model that passes tests may still fail as an assistant if its reasoning is unnatural, its terminology is inconsistent, or its explanation is contaminated by non-target scripts.

To study this gap, we present **Burmese-Coder-4B**, a Burmese coding assistant adapted from the Gemma-3 4B family through supervised fine-tuning and preference-based alignment. In parallel, we

[1] https://huggingface.co/WYNN747/burmese-coder-4b

evaluate the model with a language-aware benchmark pipeline that combines functional correctness with rubric-based assessment of Burmese fluency, instruction following, terminology quality, semantic correctness, and mixed-language penalty. Results show that Burmese-Coder-4B matches the base model on Pass@1 while substantially improving Burmese-language quality, suggesting that functional correctness alone is insufficient for evaluating low-resource coding assistants.

2 Task, Datasets, and Evaluation Framework

2.1 Task Setting

We study Burmese code generation in a setting where a model must both solve programming tasks and explain code behavior in natural Burmese. This requires maintaining a clear boundary between two distinct output regimes: natural-language reasoning, which should be expressed in Burmese, and programming syntax, which should remain standard and untranslated. In practice, this distinction is non-trivial. A response may be functionally correct at the code level while still be a poor assistant for native users if its explanation is unnatural, terminologically unstable, or contaminated by non-target scripts.

2.2 Datasets

To support this setting, we use two complementary resources. **Burmese MBPP** serves as the supervised training corpus, teaching the model how to respond to Burmese programming instructions with code and step-by-step explanations. **Burmese HumanEval** serves as the held-out benchmark for standardized functional evaluation. These datasets play distinct roles: the former supports model adaptation, while the latter supports reproducible measurement.

This distinction is particularly important in low-resource settings. A training corpus alone cannot provide a standardized benchmark, while an evaluation set alone cannot improve model behavior in the target language. Expanding both sides therefore makes it possible to study not only whether a Burmese coding model can be trained, but also whether its improvements are measurable and comparable across future systems.

2.3 Evaluation Framework

A central challenge in this problem is that conventional automatic metrics do not fully capture Burmese coding quality. Unit-test metrics such as Pass@1 measure functional correctness, but they do not determine whether an explanation is natural in Burmese, faithful to the instruction, terminologically appropriate, or contaminated by non-target scripts. We considered reference-based metrics such as BLEU and chrF, but omit them from the final evaluation because they do not adequately represent either coding quality or Burmese linguistic quality in this setting.

Accordingly, we adopt a two-track evaluation framework. The first track measures **functional correctness** through unit-test execution. The second track measures **language-aware quality** through **LLM-as-a-Judge** scoring. Rather than relying on a single opaque quality label, the judge-based track evaluates responses along multiple dimensions: fluency, instruction following, semantic correctness, terminology quality, and mixed-language penalty. Together, these dimensions capture properties that directly affect native-language usability but are not observable from functional correctness alone.

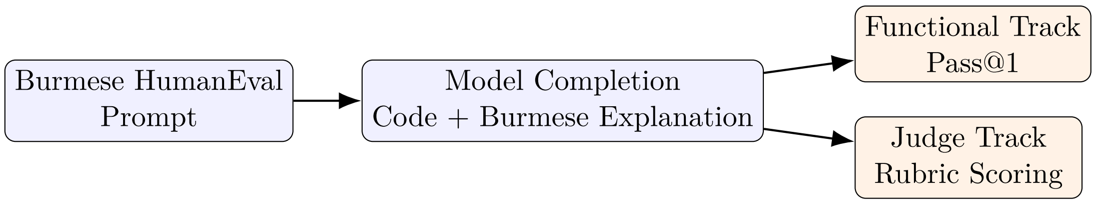

Figure 1: Evaluation flow for Burmese coding assistants. Each model completion is assessed through two complementary tracks: functional execution and rubric-based LLM judging.

To make the judge-based track interpretable, we score each response along multiple dimensions rather than using a single undifferentiated quality score. These dimensions reflect properties that matter for native Burmese users in practical coding assistance: whether the explanation reads naturally, follows the instruction, preserves the intended program logic, uses appropriate technical terminology, and avoids contamination from non-target scripts. The aggregate rubric score is computed as

$$\text{Rubric Score} = \frac{\text{Fluency} + \text{Instruction Following} + \text{Semantic Correctness} + \text{Terminology}}{4} - \text{Mix Penalty}. \tag{1}$$

This formulation treats mixed-language contamination as an explicit penalty rather than folding it into a generic quality average. As a result, the reported rubric score reflects both positive language quality and robustness against a key low-resource failure mode.

Table 1: Dimensions used in the judge-based evaluation framework. Higher is better except Mix Penalty.

Dimension	**Description**
Fluency	Measures whether the Burmese explanation is natural, readable, and well-formed for a native speaker.
Instruction Following	Evaluates whether the response faithfully follows the Burmese instruction, including both code behavior and explanation.
Semantic Correctness	Assesses whether the explanation and generated solution are logically aligned with the intended task.
Terminology	Checks whether programming concepts are expressed using appropriate and consistent Burmese technical terms.
Mix Penalty	Measures contamination by foreign-language or foreign-script tokens (lower is better).

Publicly discussed Burmese coding baselines and standardized evaluation pipelines remain scarce. This makes it difficult to compare localized Burmese coding models under a shared protocol and difficult to separate real improvements from anecdotal examples. The framework used in this work addresses this gap by pairing a Burmese adaptation of HumanEval with a reusable evaluation suite that supports unit-test evaluation and OpenAI-compatible LLM-as-a-Judge scoring through a unified workflow.

3 Model Adaptation

We adapt the Gemma-3 4B family in two stages. The first stage uses **supervised fine-tuning (SFT)** on Burmese programming instruction data to teach the model how to generate code and explain it in Burmese. The second stage applies **Direct Preference Optimization (DPO)** to reduce multilingual drift and improve linguistic stability.

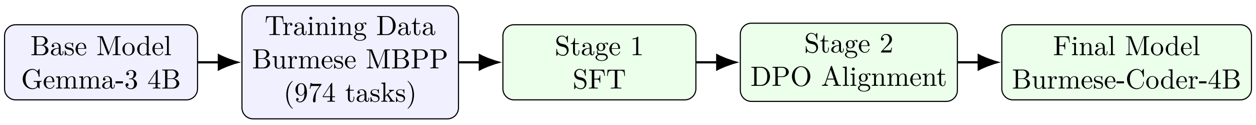

Figure 2: Two-stage adaptation pipeline for Burmese-Coder-4B. The base Gemma-3 4B model is first adapted with supervised fine-tuning on Burmese MBPP, then aligned with DPO to reduce mixed-language drift and improve Burmese response quality.

The SFT stage uses a Burmese MBPP-style dataset containing **974 programming tasks**. Each instance includes a Burmese instruction, a Python solution, and a Burmese explanation of the code logic. This stage primarily teaches the model how to structure technical responses in Burmese rather than only emit code.

The SFT implementation uses parameter-efficient fine-tuning with **LoRA**, where low-rank adapter weights are trained instead of updating the full base model. The core setup uses context length 2048, 4-bit loading, LoRA rank 16, LoRA alpha 16, batch size 16, gradient accumulation 4, learning rate 2×10^{-4}, and 15 warmup steps.

After SFT, we apply DPO to improve linguistic stability. DPO is a preference-based alignment method in which the model learns to prefer a better response over a worse response for the same prompt. In this work, the preferred response is the cleaner Burmese response, while the rejected response contains undesirable behaviors such as script mixing, hallucinated foreign tokens, or less stable technical phrasing. The DPO stage uses context length 2048, 4-bit loading, batch size 2, gradient accumulation 16, learning rate 3×10^{-6}, 300 max steps, and $\beta = 0.5$. Here, β controls how strongly the model is pushed to prefer the chosen response over the rejected one; we use a relatively strong value to make mixed-language penalties more effective.

4 Experimental Setup

We evaluate on **Burmese HumanEval**, a Burmese adaptation of HumanEval containing **100 programming problems**. Each task provides a Burmese prompt and hidden unit tests. We compare three systems: **burmese-coder-4b** (ours), **gemma3_4b** (base model), and **qwen2.5_3b**.

The evaluation framework uses two complementary tracks:

- **Functional track:** unit-test execution with Pass@1.
- **Judge track:** rubric-based scoring with an LLM judge.

The judge track measures qualities that are difficult to capture with conventional metrics alone, including Burmese fluency, instruction following, semantic correctness, terminology quality, and mixed-language penalty. Rubric scores are reported using two separate LLM judges: DeepSeek V3 and Gemini 2.5. The judge interface is OpenAI-compatible, making the framework portable across different model backends.

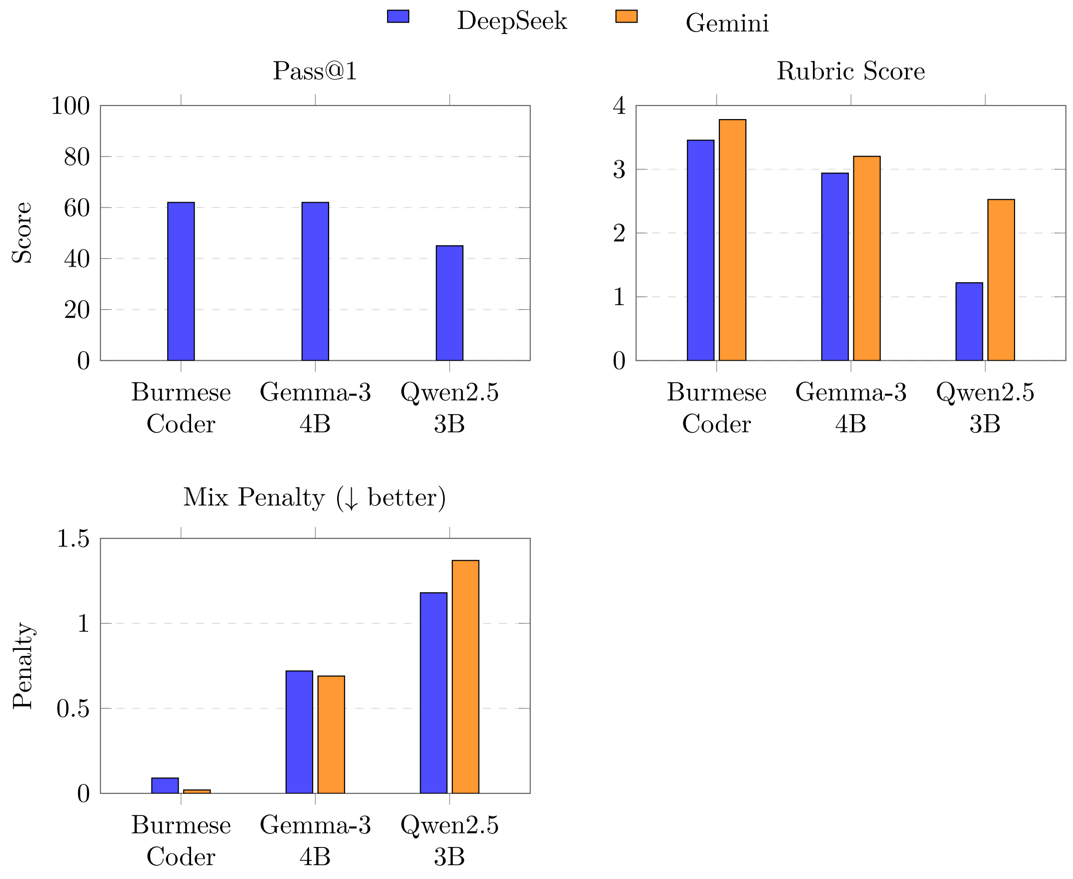

Figure 3: Benchmark results on Burmese coding evaluation. Top-left: functional correctness measured by Pass@1. Top-right: overall rubric score from two LLM judges. Bottom-left: mixed-language penalty, where lower is better. Burmese-Coder-4B matches the base Gemma-3 4B model on Pass@1 while achieving higher rubric scores and substantially lower cross-language mixing.

The end-to-end training was run on an H100 GPU on RunPod cloud infrastructure and cost approximately USD 40 over about 2.75 hours.

5 Results

Table 2 and Figure 3 present the main benchmark comparison. The key result is that Burmese-Coder-4B matches the base Gemma model on functional correctness while achieving substantially better Burmese-language quality under rubric-based evaluation. This pattern is consistent across both LLM judges and is especially visible in the mixed-language penalty, where the aligned model exhibits far less cross-script contamination.

Across both LLM judges, the most striking improvement is the reduction in mixed-language contamination. Under Gemini, the mix penalty decreases from 0.69 for the base Gemma model to 0.02 for Burmese-Coder-4B. Under DeepSeek, it decreases from 0.72 to 0.09. At the same time, the fine-tuned model improves fluency, terminology, and semantic alignment while preserving the same Pass@1 as the base model.

Table 2: Main benchmark comparison on Burmese coding evaluation. Higher is better except Mix Penalty.

Model	Pass@1 (%)	Rubric (DeepSeek)	Rubric (Gemini)
burmese-coder-4b	**62.0**	**3.456**	**3.779**
gemma3_4b	62.0	2.939	3.203
qwen2.5_3b	45.0	1.220	2.526

Table 3: Detailed rubric breakdown under DeepSeek judge. Higher is better except Mix Penalty.

Model	Fluency	Instr.	Semantic	Terminology	Mix Penalty
burmese-coder-4b	**3.82**	**3.43**	**3.26**	**3.62**	**0.09**
gemma3_4b	3.52	3.26	2.95	3.18	0.72
qwen2.5_3b	1.75	1.28	1.16	1.45	1.18

6 Analysis

The results suggest that the primary benefit of adaptation is not a large increase in raw code-solving ability, but a substantial improvement in linguistic reliability. Supervised fine-tuning improves the model's ability to structure Burmese explanations and follow Burmese instructions, while preference-based alignment appears to be the main driver of reduced mixed-script hallucination. This explains why the fine-tuned model can remain comparable on Pass@1 yet feel significantly better for native users.

More broadly, these findings indicate that functional correctness alone can mask important usability differences in low-resource coding assistants. In Burmese, a model can solve the task while still failing at the explanation layer through unstable terminology, unnatural phrasing, or multilingual drift. Language-aware evaluation is therefore not an auxiliary feature, but a necessary part of the measurement protocol.

A representative qualitative example further supports this pattern. When asked to organize files into folders, the untuned Gemma response mixed Burmese with fragments from other writing systems in both the explanatory text and surrounding instruction. By contrast, Burmese-Coder-4B produced a cleaner and more consistent response while keeping the code in standard Python. This qualitative difference aligns with the large drop in mix penalty observed in the benchmark tables.

7 Limitations

This study has several limitations. First, the training dataset is still relatively small for code modeling. Second, evaluation is centered on Python and HumanEval-style tasks rather than broader software engineering workflows such as repositories, debugging, tool use, or long-horizon coding assistance. Third, rubric-based evaluation depends on judge models, which may themselves introduce bias or variance. Finally, while the results indicate reduced multilingual drift, the exact token-level causes of this phenomenon remain an open research question.

8 Conclusion

We introduced Burmese-Coder-4B, a compact Burmese coding assistant trained with supervised fine-tuning and preference-based alignment. Although the model does not outperform the base

Table 4: Detailed rubric breakdown under Gemini judge. Higher is better except Mix Penalty.

Model	Fluency	Instr.	Semantic	Terminology	Mix Penalty
burmese-coder-4b	**3.98**	**3.78**	**3.55**	**3.92**	**0.02**
gemma3_4b	3.71	3.66	3.16	3.49	0.69
qwen2.5_3b	2.40	2.41	3.08	3.19	1.37

Gemma model on Pass@1, it produces substantially better Burmese-language explanations, better terminology usage, and far lower mixed-language contamination. These findings show that localized coding assistants should be evaluated not only by code correctness, but also by how well they communicate with users in the target language.

References

[1] Mark Chen et al. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*, 2021.

[2] Wai Yan Nyein Naing. Burmese HumanEval: A Benchmark for Evaluating Burmese Coding Assistants. Hugging Face dataset, 2026. https://huggingface.co/datasets/WYNN747/burmese-human-eval

[3] Wai Yan Nyein Naing. Burmese MBPP: A Large-Scale Programming Dataset for Burmese Coding Assistants. Hugging Face dataset, 2026.

[4] Wai Yan Nyein Naing. Burmese-Coding-Eval: A Translated and Expanded HumanEval Benchmark for Burmese Programming Assistants. GitHub repository, 2026. https://github.com/WaiYanNyeinNaing/burmese-coding-eval

[5] Rafael Rafailov et al. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. *arXiv preprint arXiv:2305.18290*, 2023.