PyBhasha at BLP-2025 Task 2: Effectiveness of Semantic-Aware Translation and Ensembling in Bangla Code Generation

Foyez Ahmed Dewan*, **Nahid Montasir Rifat***
Department of Computer Science and Engineering
Rajshahi University of Engineering & Technology, Rajshahi, Bangladesh
foyez.ruet767@gmail.com, nahidmuntasir2@gmail.com

Abstract

In this paper, we present our submission to Task 2 of the BLP-2025 shared task on code generation from Bangla instructions. Our approach focused on enhancing instruction quality through translation and improving model performance with a two-stage ensemble strategy. We evaluated two proprietary and several open-source models under three instruction settings: original Bangla instructions, Bangla instructions translated into English using Facebook NLLB, and instructions rewritten in English with GPT-4.1. Experimental results showed that GPT-4.1-rewritten instructions consistently achieved the highest accuracy across models. For final predictions, we used a two-stage ensemble, achieving a *pass@1* score of **80.0%** on the hidden test set and securing 12th place on the official leaderboard. Additionally, we conducted a qualitative analysis of selected translations to illustrate how variations in instruction phrasing influenced model outputs.

1 Introduction

Code generation is the task of creating computer programs automatically from natural language descriptions. It allows users to write instructions in plain language and have a model produce the corresponding executable code. Beyond practical applications, it has emerged as a key benchmark for evaluating the reasoning and problem-solving performance of large language models (LLMs). Recent systems such as Codex, CodeLLaMA, and DeepSeek-Coder have achieved strong results on several English-centric programming benchmarks, approaching state-of-the-art performance under favorable conditions (Rozière et al., 2024; Guo et al., 2024). Yet, this success does not extend uniformly to low-resource languages like Bangla, where code generation remains underexplored due to limited training data, inconsistent instruction formats, and unreliable translation quality. Consequently, models often misinterpret Bangla prompts, producing syntactic errors or semantically incorrect code.

To address these challenges, we participate in Task 2 of the BLP-2025 shared task, which focuses on generating Python code from Bangla instructions. Our work investigates the role of instruction formulation in improving code generation quality. Specifically, we compare three input styles: raw Bangla instructions, English translations generated by Facebook NLLB, and refined English instructions rewritten by GPT-4.1. This analysis highlights how translation quality and semantic clarity directly affect model performance. While instruction reformulation substantially improves model outputs, we also explore a simple two-stage ensemble strategy to further enhance accuracy and robustness.

In summary, the main contributions of this work can be outlined as follows:

- We provide a performance comparison of proprietary, multilingual, and Bangla-centric LLMs in a one-shot code generation setting.
- We demonstrate that high-quality English reformulations of Bangla instructions substantially improve code generation outcomes.
- We introduce a simple yet effective two-stage ensemble strategy that achieves higher accuracy than standalone models.

2 Related Work

Recent Bangla-centric work such as TigerCoder (Raihan et al., 2025b) and BongLLaMA (Zehady et al., 2024) demonstrates that targeted Bangla instruction datasets and language-specific fine-tuning can improve Bangla text-to-code generation. While these studies highlight the potential of multilingual and Bangla-centric models, performance still lags

*Equal contribution.

Proceedings of the Second Workshop on Bangla Language Processing (BLP-2025), pages 624–628

behind English benchmarks, leaving room for further exploration of strategies to advance Bangla code generation. Iskander et al. (2024) show that dataset quality has a major effect on LLM performance: noisy or misaligned instruction–code pairs degrade results, while filtered and curated subsets lead to significant gains even with less data, underscoring the importance of high-quality training data in low-resource scenarios. Vahtola et al. (2025) demonstrate that GPT-4 is highly effective at paraphrasing and following linguistic instructions, making it well suited for producing semantically faithful reformulations in translation-based workflows where clarity and fidelity are essential. Brown et al. (2020) further show that in-context learning improves code generation, with few-shot setups often outperforming one-shot and zero-shot; however, one-shot prompting remains widely used because it provides minimal context while ensuring consistency across models, making it suitable for controlled comparisons. Ensembling has also been explored in code generation, with systems like AlphaCode boosting pass rates through candidate reranking (Li et al., 2022). However, lightweight fallback ensembles remain largely unexplored in low-resource contexts such as Bangla. Building on these findings, our work investigates how instruction reformulation and ensemble strategies can improve Bangla code generation under a one-shot setup, addressing both the challenges of translation quality and the limitations of low-resource settings.

3 Task Description

The primary objective of this shared task (Raihan et al., 2025c) is to develop systems capable of generating Python code from Bangla programming instructions. Participating models must synthesize accurate Python solutions that satisfy the corresponding unit tests. Evaluation is performed strictly based on whether the generated code passes all test cases. By linking Bangla-language instructions to executable Python programs, this task advances research in multilingual code generation and contributes to building robust systems for low-resource languages.

3.1 Dataset Description

The dataset for this task was released in three splits: trial set, dev set, and test set. The sample distribution is summarized in Table 1. Each sample consists of a Bangla instruction describing a programming problem. The trial set additionally provides reference solutions together with the complete set of unit tests. In contrast, neither the dev set nor the test set contains reference solutions. The dev set includes the full unit test suite, while the test set offers only a single visible test case, with the remaining cases kept hidden for final evaluation.

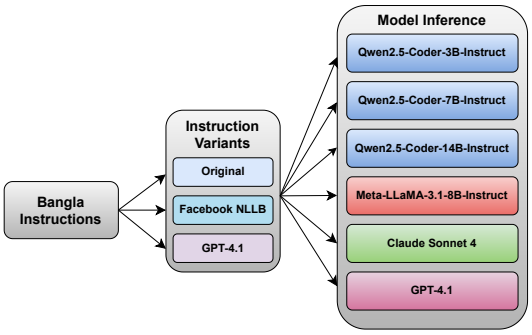

Figure 1: Model inference pipeline using three instruction styles (Original Bangla, Facebook NLLB, GPT-4.1) across multiple models.

Data Splits	Total Samples
trial	74
dev	400
test	500

Table 1: Overview of the Task 2 dataset splits.

The dev and test sets use a subset of Bangla-translated versions of the original English mHumanEval and MBPP datasets, as introduced in (Raihan et al., 2025a,b).

4 Experiments

We conducted a series of experiments to identify effective strategies for Bangla-to-Python code generation. All experiments were carried out on Kaggle using an NVIDIA P100 GPU with 16 GB memory. To enable efficient evaluation of large models (up to 14B parameters) within this limited hardware budget, we performed inference using 4-bit quantization through the `bitsandbytes` library. The subsequent subsections detail the model selection process, the instruction variants we explored, and our proposed ensemble approach.

4.1 Model Selection

We evaluated a set of open-source and proprietary LLMs on the dev set and test set under one-shot prompting, as presented in Figure 1 . One-

Model Name	Dev Phase			Test Phase		
	Original	NLLB	GPT-4.1	Original	NLLB	GPT-4.1
TigerLLM-9B-it	72.5	65.5	77.3	58.6	56.8	63.6
Qwen2.5-Coder-3B-Instruct	52.0	57.0	67.0	47.0	48.4	56.2
Qwen2.5-Coder-7B-Instruct	62.0	66.0	73.0	59.8	64.4	69.2
Qwen2.5-Coder-14B-Instruct	77.0	76.0	82.0	67.2	68.2	76.0
Meta-LLaMA-3.1-8B-Instruct	48.0	50.0	56.0	45.6	47.2	52.0
Claude Sonnet 4	–	–	82	–	–	72.4
GPT-4.1	–	–	77.8	–	–	71.0
Two-Stage Ensemble (Qwen2.5-Coder-14B-Instruct + Claude Sonnet 4)	–	–	–	–	–	**80.0**

Table 2: *pass@1* scores across three instruction variants (Original Bangla, Facebook NLLB translation, and GPT-4.1 rewriting) in both dev and test phases.

shot prompting was selected to provide minimal context while ensuring consistency across models, thereby enabling a fair comparison of their code generation capabilities. The models considered included Bangla-centric models such as `TigerLLM-9B-it`, and multilingual models including `Qwen2.5-Coder-3B/7B/14B-Instruct` (Hui et al., 2024), `Meta-Llama-3.1-8B-Instruct`, as well as proprietary models `Claude Sonnet 4` and `GPT-4.1`. This evaluation allowed us to compare the effectiveness of models specifically tailored for Bangla against those trained on broader multilingual corpora.

4.2 Instruction Variants

To investigate the effect of instruction formulation on model performance, we experimented with three variants of the input instructions during the dev phase: (i) the original Bangla instructions provided in the dataset, (ii) English translations generated using the Facebook NLLB translation model, and (iii) English instructions rewritten from Bangla using GPT-4.1. For fairness, all models were evaluated under the same experimental setup across these three variants. This design enabled us to gain a deeper understanding of how different instruction styles influenced code generation quality.

4.3 Ensemble Approach

To maximize code generation success, we employed a two-stage ensemble strategy using the best-performing models from the dev phase, as illustrated in Figure 2. `Qwen2.5-Coder-14B-Instruct` was used as the primary model to generate Python code for all instructions. Any samples that failed their unit tests were then re-generated by `Claude Sonnet 4`, enabling the secondary model to recover from errors made by the primary. This sample-level ensemble leveraged complementary strengths to increase the overall success rate.

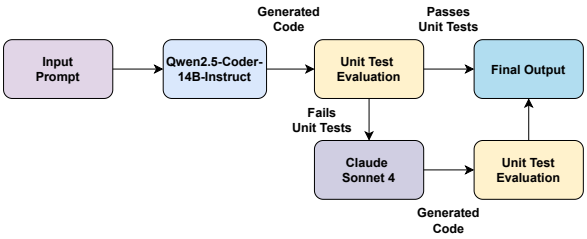

Figure 2: Two-stage ensemble strategy: The primary model (Qwen2.5-Coder-14B-Instruct) generates a code response which is evaluated via unit tests. If the code fails, the same prompt is passed to a secondary model (Claude Sonnet 4) for regeneration.

5 Results and Analysis

5.1 Initial Evaluation

We evaluated all models using the **pass rate** (*pass@1*) metric. Each model was tested in three types of instructions as described in Section 4.2

Results are reported on both dev sets and test sets, with evaluations performed against complete unit test suites. As shown in Table 2, among proprietary models, `Claude Sonnet 4` achieved the highest accuracy on both the dev and test sets when paired with GPT-4.1-translated instructions. Similarly, among the open-source models, `Qwen2.5-Coder-14B-Instruct` consistently outperformed others, even surpassing proprietary models. This highlights the strong generalization capability of Qwen when paired with high-quality instruction translations.

Other open-source models, such as `Meta-Llama-3.1-8B-Instruct` and `TigerLLM-9B-it`, also showed marked im-

id	Bangla Text	Facebook NLLB	GPT-4.1
33	প্রদত্ত অ্যারেতে সমস্ত সংখ্যার জোড়ার xor এর যোগফল খুঁজে পেতে একটি পাইথন ফাংশন লিখুন।	Write a Python function to find the sum of all the numbers in the given array.	Write a Python function to find the sum of the XOR of all pairs of numbers in a given array.
131	একটি শঙ্কুর পার্শ্বীয় পৃষ্ঠতল এলাকা খুঁজে পেতে একটি ফাংশন লিখুন।	Write a function to find a cornered side area.	Write a function to find the lateral surface area of a cone.

Figure 3: Examples of translation quality, showing how GPT-4.1 preserves semantic intent more accurately than Facebook NLLB for complex Bangla instructions.

provements when paired with GPT-4.1-based instructions, underscoring the importance of clear and semantically rich prompts in instruction-to-code generation tasks.

5.2 Ensemble Approach Outcome

We leveraged the two-stage ensemble strategy described in Section 4.3 to further boost performance. The ensemble approach achieved a **pass rate** (*pass@1*) of **80.0%** on the hidden test set under full unit test evaluation. This score represents a measurable improvement over the standalone performance of `Qwen2.5-Coder-14B-Instruct` (76.0%) and `Claude Sonnet 4` (72.4%). By selectively routing failed generations from Qwen to Claude, the ensemble effectively recovered additional correct outputs, confirming its utility in minimizing failure cases.

5.3 Impact of Instruction Formulation

To better understand why instructions translated via GPT-4.1 consistently outperformed others, we manually examined a few representative translations to assess how accurately and clearly each model conveyed the original instruction's meaning. Two such examples from the dev set are presented in Figure 3.

These examples demonstrate that GPT-4.1 translations more faithfully preserve semantic precision than those produced by Facebook NLLB, particularly for mathematically or structurally complex tasks. For instance, GPT-4.1 correctly interpreted the Bangla term "পার্শ্বীয় পৃষ্ঠতল" as "lateral surface area," whereas NLLB rendered it as "cornered side area," a phrase lacking geometric validity. Similarly, in a case involving the XOR operation, GPT-4.1 accurately preserved the intent to compute the sum of XORs over all pairs, while NLLB reduced the instruction to a basic summation task.

Bangla-native models such as `TigerLLM` achieved stronger performance when paired with either the original Bangla instructions or GPT-4.1-translated English, likely due to the increased semantic richness and clarity. In contrast, NLLB-translated prompts introduced ambiguity, leading to degraded outcomes. For all other models (e.g., `Qwen`, `LLaMA`), we observed a consistent performance ranking: GPT-4.1 > NLLB > Original Bangla, reflecting their English-dominant training distributions.

These findings suggest that, beyond simple language alignment, the semantic clarity and task specificity of instructions are critical for improving code generation quality.

6 Conclusion

In this work, we explored the impact of instruction translation on Bangla code generation tasks, with a focus on translation quality and ensemble modeling. Our evaluation demonstrated that GPT-4.1-translated instructions substantially improved performance compared to both raw Bangla and NLLB-generated instructions. We also conducted qualitative analyses to explain model sensitivities to different instruction styles. Future work will focus on fine-tuning Bangla-native models using high-quality synthetic prompts to further enhance generalization.

Limitations

At the time of our experiments, the GPT-5 API had not yet been publicly released; therefore, all instruction-rewriting experiments were conducted using GPT-4.1. While rewriting instructions in English with GPT-4.1 yielded clear improvements, our approach was limited by the absence of fine-

tuning on Bangla-specific data. We expect that performance could be further enhanced through targeted fine-tuning of Bangla-centric models, as well as multilingual models trained directly on Bangla instructions. More broadly, our work highlights the lack of large, high-quality Bangla code-generation datasets; addressing this gap through dataset creation and model adaptation remains an important direction for future research.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *Preprint*, arXiv:2401.14196.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, and 5 others. 2024. Qwen2.5-coder technical report. *Preprint*, arXiv:2409.12186.

Shadi Iskander, Sofia Tolmach, Ori Shapira, Nachshon Cohen, and Zohar Karnin. 2024. Quality matters: Evaluating synthetic data for tool-using LLMs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4958–4976, Miami, Florida, USA. Association for Computational Linguistics.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, and 7 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Nishat Raihan, Antonios Anastasopoulos, and Marcos Zampieri. 2025a. mHumanEval - a multilingual benchmark to evaluate large language models for code generation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11432–11461, Albuquerque, New Mexico. Association for Computational Linguistics.

Nishat Raihan, Antonios Anastasopoulos, and Marcos Zampieri. 2025b. Tigercoder: A novel suite of llms for code generation in bangla. *Preprint*, arXiv:2509.09101.

Nishat Raihan, Mohammad Anas Jawad, Md Mezbaur Rahman, Noshin Ulfat, Pranav Gupta, Mehrab Mustafy Rahman, Shubhra Kanti Karmakar, and Marcos Zampieri. 2025c. Overview of BLP-2025 task 2: Code generation in bangla. In *Proceedings of the Second Workshop on Bangla Language Processing (BLP-2025)*. Association for Computational Linguistics (ACL).

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, and 7 others. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Teemu Vahtola, Songbo Hu, Mathias Creutz, Ivan Vulić, Anna Korhonen, and Jörg Tiedemann. 2025. Analyzing the effect of linguistic instructions on paraphrase generation. In *Proceedings of the Joint 25th Nordic Conference on Computational Linguistics and 11th Baltic Conference on Human Language Technologies (NoDaLiDa/Baltic-HLT 2025)*, pages 755–766, Tallinn, Estonia. University of Tartu Library.

Abdullah Khan Zehady, Safi Al Mamun, Naymul Islam, and Santu Karmaker. 2024. Bongllama: Llama for bangla language. *Preprint*, arXiv:2410.21200.