Musafir at BLP_2025 Task 2: Generating Python Code from Bangla Prompts using a Multi-Model Cascade and Unit Test Validation

Sakibul Hasan[1], **Md Tasin Abdullah**[1], **Abdullah Al Mahmud**[1], **Ayesha Banu**[1]

[1]Department of Computer Science and Engineering,
Chittagong University of Engineering & Technology (CUET)

Correspondence:
u2004043@student.cuet.ac.bd, u2004059@student.cuet.ac.bd,
u2004057@student.cuet.ac.bd, ayesha.banu@cuet.ac.bd

Abstract

This paper presents our approach for the BLP25 Task 2: Code Generation in Bangla. To address the scarcity of Bangla–code training data, we adopt a two-stage pipeline. First, Bangla problem statements are translated into English using a neural translation model optimized for preserving technical semantics. Then, the translated text is passed to a Qwen-based code generation model to produce executable solutions. This translation–generation strategy leverages the strengths of English-centric code models while ensuring fidelity to the original Bangla instructions. Our system achieved competitive performance on the leaderboard, achieving the **3rd place** with a score of **91.8%** while demonstrating that translation-augmented pipelines are effective for low-resource code generation tasks.

Keywords: Large Language Model, Qwen, Unit Test.

1 Introduction

Natural language to code generation aims to translate human-readable descriptions into syntactically correct and semantically meaningful code. While notable advances have been achieved in English with large datasets and pretrained models, low-resource languages like Bangla remain underexplored. This limitation prevents native Bangla speakers from fully benefiting, as most existing code generation systems are not designed for Bangla instructions.

Bangla holds immense potential for broader accessibility in computational tasks. (Nahin et al., 2025) introduce TituLLMs, a family of Bangla LLMs with extensive benchmarking across diverse tasks. (Bhattacharjee et al., 2022) propose BanglaBERT, a pretrained language model and benchmark suite for low-resource Bangla understanding. (Kowsher et al., 2022) develop a transformer-based Bangla-BERT optimized for transfer learning and efficient Bangla language understanding. However, the lack of large, parallel Bangla–code datasets hinders the direct training and evaluation of Bangla code generation models. This gap shows the need for new strategies to adapt code generation systems to Bangla without large language-specific resources.

In this work, we propose a two-stage pipeline to address Bangla-to-code generation. First, Bangla problem descriptions are translated into English, enabling the use of pretrained English-centric code models. The translated instructions are then processed by a Qwen-based code generation model to produce executable solutions. This translation–generation framework mitigates the scarcity of Bangla–code data while leveraging multilingual modeling advances. Our system shows competitive performance, demonstrating translation as an effective bridge for low-resource code generation.

2 Related Work

Bangla is the seventh most spoken language in the world, yet it remains underrepresented in LLM research, particularly in code generation. The lack of Bangla–code parallel data has hindered systems that can natively process Bangla instructions. This gap has led to several recent initiatives to create Bangla-specific benchmarks and LLMs. (Yang et al., 2024) introduce Qwen2, the next-generation models in the Qwen family, ranging from 0.5B to 72B parameters in both dense and Mixture-of-Experts (MoE) variants. The report highlights improvements in multilingual coverage, reasoning ability, and training efficiency, with strong benchmark results across general language understanding, coding, and instruction-following tasks. Earlier, (Bai et al., 2023) presented the first Qwen technical report, outlining the development of large-scale multilingual models with specialized variants for chat, coding, and mathematics.

Their work established Qwen as a versatile family of open-source LLMs, demonstrating competitive performance against contemporaneous models and paving the way for subsequent expansions such as Qwen2.

(Raihan et al., 2025a) and BLP-2025 Task 2 (Raihan et al., 2025c) provide systematic benchmarks for evaluating Bangla code generation, revealing that existing multilingual models perform poorly on Bangla. Complementing these efforts, TigerCoder (Raihan et al., 2025b) introduces Bangla-specialized LLMs fine-tuned for code generation using a large instruction–code dataset built through translation and synthetic generation. Trained on the MBPP-Bangla benchmark, TigerCoder models outperform multilingual baselines, achieving state-of-the-art results in Bangla code generation. Our approach introduces a two-stage translation–generation pipeline where Bangla problem statements are translated into English and processed by a Qwen-based code model to generate Python solutions. Unlike TituLLMs and TigerCoder which trains Bangla-specialized LLMs on instruction–code datasets, our method leverages powerful English-centric models through translation to overcome data scarcity. Based on these benchmarks and models, we built our overall pipeline, shown in Figure 1, to effectively tackle Bangla-to-code generation.

3 Methodology

3.1 Dataset Description

Table 1: Dataset distribution across phases

Phase	No. of Samples in Dataset
Development phase	400
Testing phase	500

Each entry includes: Bangla natural language programming task and a set of assert statements or unit test cases used for designing, testing, and iteratively refining the pipeline.

3.2 Bangla to English Translation

The process for generating code from non-English instructions is illustrated in the accompanying figure 3. It begins with a Bangla Instruction, which is the input prompt. This instruction is processed by the Model Preparation stage, utilizing the gemma-3-12bit-unsloth-bnb model. In our approach, gemma-3-12bit-unsloth-bnb was used in a zero-shot inference setup with a custom prompt template, not fine-tuned. It served as a translation model to convert Bangla instructions into English while preserving technical semantics. Crucially, the model is provided with a Prompt Template to enforce the desired function structure and signature. Finally, the model's raw output is passed through the Translation stage, which decodes the model's output to produce the final, executable Python function. In short, the methodology shows how a language-specific prompt is combined with a structural template and processed by the Gemma model to perform a focused code generation task.

Table 2: Comparison of Bangla and English translations with verdicts.

Bangla Translation	English Translation	Verdict
একটি ত্রিভুজের পরিসীমা বের করার প্রোগ্রাম লিখ	Write a program to find perimeter of a triangle	Correct
একটি স্ট্রিং এর দীর্ঘতম পরবর্তী সাধারণ ক্রম বের করার প্রোগ্রাম লিখ	Write a program to find longest common string	Incorrect

3.3 LLM Inference & Code Generation

The translated English instruction, along with a specified code Prompt Template (defining the function signature), is fed into the Qwen 2.5-14B-Instruct model. This 14.7-billion-parameter LLM then performs the final, zero-shot code inference, producing the required Python function.

3.4 Unit Testing

The unit testing framework follows a structured workflow to ensure the correctness and reliability of generated code. As illustrated in Figure 4, the process begins with the Input stage, where predefined test cases are supplied as benchmarks for evaluating the code. These test cases are designed to cover a range of scenarios and edge cases, ensuring comprehensive assessment. In the Execution phase, the code is run within a dedicated, isolated environment, which guarantees secure and independent execution without interference from external processes. During the Testing stage, the outputs are parsed, validated, and compared against the expected results. The Evaluation phase further assesses consistency, correctness, and adherence to functional requirements, highlighting any deviations or unexpected behavior. Finally, the Decision stage categorizes each result as pass or fail, indicating whether the code satisfies the predefined

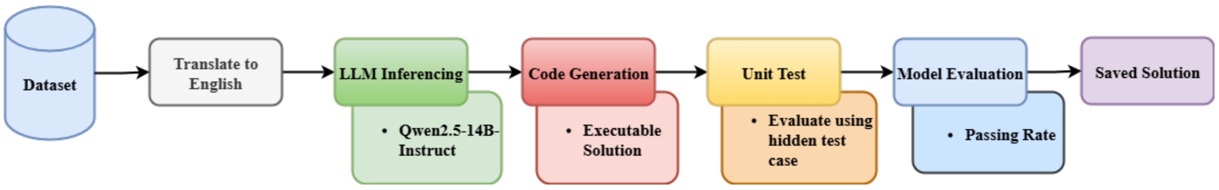


Figure 1: The overall pipeline describing the whole methodology covering the translation process to code generation with unit testing and model evaluation.

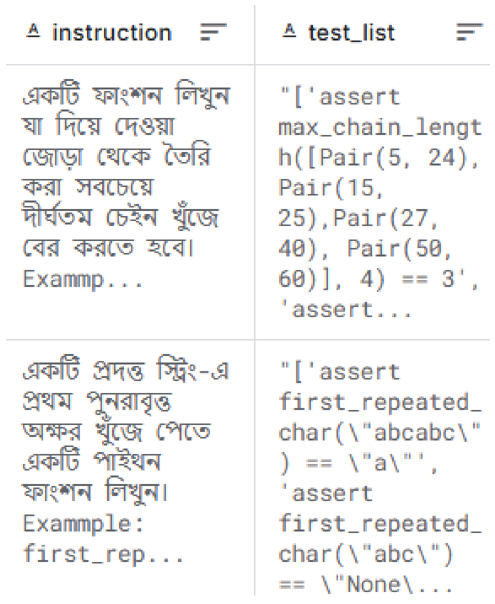
Table 3: Bangla and English prompts with corresponding code examples.

Bangla Instruction	English Instruction	Code	Result

একটি প্রদত্ত ম্যাট্রিক্সকে তার সারিগুলির যোগফল অনুযায়ী ক্রমবর্ধমান ক্রমে সাজানোর জন্য একটি ফাংশন লিখুন।	Write a function to sort a given matrix in ascending order based on the sum of its rows.	def sort_matrix(M):
		    return sorted(M, key=sum)
			Pass

একটি ত্রিভুজাকার প্রিজমের আয়তন খুঁজে বের করার জন্য একটি পাইথন ফাংশন লিখুন।	Write a Python function to find the volume of a triangular prism.	def find_Volume(l, b, h):
		    base_area = (l * b) / 2
		    volume = base_area * h
		    return volume
			Pass

প্রদত্ত সংখ্যাটি কাঠের বল কিনা তা পরীক্ষা করার জন্য একটি ফাংশন লিখুন।	Write a function to check if a given number is a Woodall number.	def is_woodall(x):
		    if x < 1:
		        return False
		    n = 1
		    while (2 ** n - 1) * n <= x:
		        if (2 ** n - 1) * n == x:
		            return True
		        n += 1
		    return False
			Fail



instruction	test_list
একটি ফাংশন লিখুন যা দিয়ে দেওয়া জোড়া থেকে তৈরি করা সবচেয়ে দীর্ঘতম চেইন খুঁজে বের করতে হবে। Exammp...	"['assert max_chain_length([Pair(5, 24), Pair(15, 25),Pair(27, 40), Pair(50, 60)], 4) == 3', 'assert...
একটি প্রদত্ত স্ট্রিং-এ প্রথম পুনরাবৃত্ত অক্ষর খুঁজে পেতে একটি পাইথন ফাংশন লিখুন। Exammple: first_rep...	"['assert first_repeated_char(\"abcabc\") == \"a\"', 'assert first_repeated_char(\"abc\") == \"None\...

Figure 2: Dataset Used in Development Phase

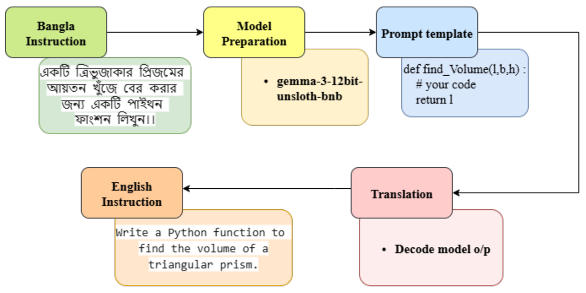

Bangla Instruction
একটি ত্রিভুজাকার প্রিজমের আয়তন খুঁজে বের করার জন্য একটি পাইথন ফাংশন লিখুন।।
Model Preparation
gemma-3-12bit-unsloth-bnb
Prompt template
def find_Volume(l,b,h) :
# your code
return 1
Translation
Decode model o/p
English Instruction
Write a Python function to find the volume of a triangular prism.

Figure 3: Bangla to English translation using gemma-3-12bit-unsloth-bnb model to generate the prompt in English.

requirements. Passing outcomes confirm that the solution meets the expected behavior, while failures reveal discrepancies that require correction. This structured approach provides a robust mechanism for automated verification, enabling reliable benchmarking of generated code in low-resource language scenarios.

4 Experiment & Analysis

There are two evaluation metrics for the task:

1. **Pass@1 (Passing rate)-** Number of hidden test cases that pass.

2. **Tie-breaker-** Shorter mean solution length.

Table 4: Comparison of **Pass@1** on different prompting methods between some Open-Source LLMs and our proposed solution.

Model	**Zero-Shot**	**Few-Shot**	**Instruction-Tuned**
Qwen2.5-Coder-14B-Instruct	82.7%	79.2%	81.4%
codegemma-7bit	76.8%	75.1%	75.7%
gpt-oss-20b	78.4%	76.9%	77.8%
Phi-4-reasoning	71.7%	67.8%	70.6%
CodeLlama-13b-Python	74.8%	72.3%	73.7%
Qwen2.5-14B + codegemma-7bit + gpt-oss-20B + phi-4-14B	**91.5%**	**88.5%**	**90.4%**

Table 5: Comparative Analysis of LLM Models: With vs Without Translation.

Model	**With Translation (%)**	**Without Translation (%)**
Combined	**91.5**	**87.5**
Qwen2.5-Coder-14B-Instruct	82.7	80.0
codegemma-7bit	78.4	75.0
Phi-4-reasoning	76.8	73.0

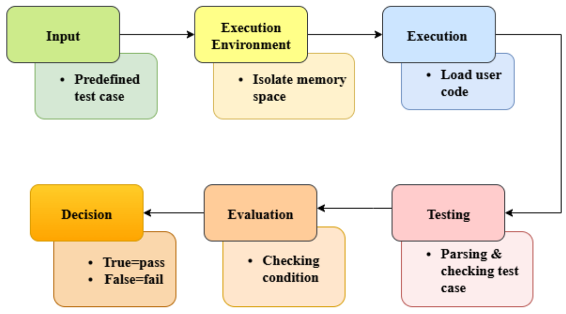


Figure 4: Unit testing framework to ensure the correctness and reliability of the code.

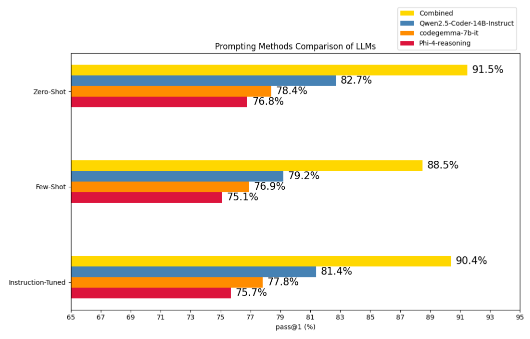


Figure 5: Comparative analysis on different prompting methods across models.

The table 4 compares the pass@1 of different models across three configurations: Zero-Shot, Few-Shot, and Instruction-Tuned. Among them Qwen2.5-14B, codegemma-7b, gpt-oss-20B, and phi-4-14B, achieves the highest performance, with an impressive 91.5% in Zero-Shot, 88.5% in Few-Shot, and 90.4% in Instruction-Tuned, demonstrating the effectiveness of the multi-model approach.

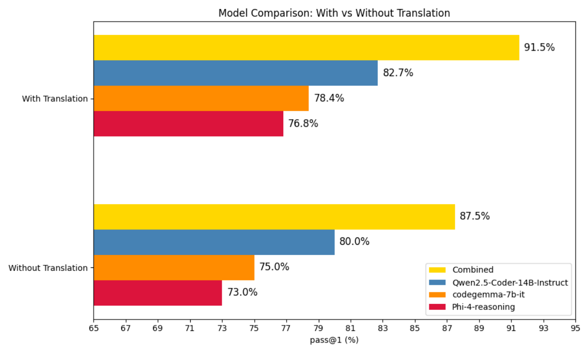


Figure 6: Comparative Analysis on Basis of Translation across Models.

The table 5 presents a comparative analysis of various LLM models with and without translation. The Combined model achieves the highest performance with 91.5% using translation, compared to 87.5% without it, showcasing benefits of translation. The results are also graphically represented in figure 5 & 6.

5 Conclusion

The paper presents an effective two-stage translation-generation pipeline to address the challenge of low-resource code generation in Bangla for the BLP25 Task 2. By first translating Bangla problem statements into English using a model optimized for technical semantics , and then passing the translated text to an English-centric Qwen-based code generation model , the approach successfully leverages the strengths of existing, powerful models designed for English. This strategy mitigates the scarcity of large, parallel

Bangla-code datasets and demonstrates that a translation-augmented framework is competitive for these tasks, achieving 3rd place with a score of 91.8% on the leaderboard.

Limitations

The proposed two-stage translation–generation pipeline shows promise for Bangla code generation but has limitations. It relies on English-centric models, which may miss technical nuances in Bangla prompts, and translation errors can affect output quality. Evaluation is limited to specific tasks, so testing on more diverse, real-world problems is needed to assess generalizability. [1]

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, and 1 others. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Abhik Bhattacharjee, Tahmid Hasan, Wasi Uddin Ahmad, Kazi Samin Mubasshir, Md Saiful Islam, Anindya Iqbal, M. Sohel Rahman, and Rifat Shahriyar. 2022. Banglabert: Language model pretraining and benchmarks for low□resource language understanding in bangla. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 1318–1327.

M. Kowsher, Abdullah As Sami, Nusrat Jahan Prottasha, Mohammad Shamsul Arefin, Pranab Kumar Dhar, and Takeshi Koshiba. 2022. Bangla-bert: Transformer-based efficient model for transfer learning and language understanding. *IEEE Access*, 10:91855–91870.

Shahriar Kabir Nahin, Rabindra Nath Nandi, Sagor Sarker, Quazi Sarwar Muhtaseem, Md Kowsher, Apu Chandraw Shill, Md Ibrahim, Mehadi Hasan Menon, Tareq Al Muntasir, and Firoj Alam. 2025. Titullms: A family of bangla llms with comprehensive benchmarking. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24922–24940, Vienna, Austria.

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. *arXiv 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).

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, and 1 others. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

[1]Code is available here