Barrier Breakers at BLP-2025 Task 2: Enhancing LLM Code Generation Capabilities through Test-Driven Development and Code Interpreter

Sajed Jalil
Apollo Applications Group
Reston, VA, USA
sajed@apolloapps.ai

Shuvo Saha
University of Dhaka
Dhaka, Bangladesh
bsse0705@iit.du.ac.bd

Hossain Mohammad Seym
Islamic University of Technology
Dhaka, Bangladesh
hossainseym@iut-dhaka.edu

Abstract

Over the past few years, improving LLM code generation capabilities has been a key focus in NLP research. Despite Bengali having 242 million native speakers worldwide, it receives little attention when it comes to training LLMs. More recently, various fine-tuning and augmented generation techniques have been employed to significantly enhance code generation performance. However, they require considerable expertise and resources to utilize effectively as an end user. The goal of our work is to democratize access to powerful code generation tools in resource-constrained emerging markets, enabling users to leverage them in their native language.

We introduce a novel approach that combines Test-Driven Development (TDD) and Code Interpreter (CI), utilizing open-weight models, which improves the baseline accuracy for code generation with Bengali prompts and achieves an overall accuracy of **85%**. Our approach requires no finetuning and proves that even the smallest models in the same family can attain up to **98%** accuracy compared to the largest models. All of our results [1] are publicly shared in GitHub for validation and reproducibility.

1 Introduction

Large Language Models (LLMs) have gained significant attention across various research communities since the release of ChatGPT in 2022 [2]. Initially known as generalized text completion models, LLMs quickly found their way into more specialized tasks such as code, image, and audio generation. Specifically, the impact is visible in the code generation domain. There has been a significant transformation in the daily workflow of the software engineers with these models (Jalil, 2025).

Despite being the 5th most spoken language worldwide, Bengali is not included in most of the top models as a primary language for training data (Raihan et al., 2025b). Even in cross-lingual settings, most models tend to reflect Western perspectives (Myung et al., 2024). Additionally, prior studies have demonstrated that multilingual tokenizers are often inefficient and require additional resources during training (Ali et al., 2024).

With these constraints in mind, we propose our work on enhancing existing open-weight LLMs of various sizes by combining Test-Driven Development (TDD) and Code Interpreter (CI) without the need for fine-tuning. In this shared task with Bengali prompts, we investigate the following research questions that are crucial for advancing the field of multilingual code generation -

RQ1: *How far can performance improve without fine-tuning or external data augmentation?*

RQ2: *Can smaller models approach larger model performance?*

RQ3: *What approach is most effective in improving vanilla (baseline) LLM accuracy?*

RQ4: *To what extent do these approaches reduce compilation errors?*

2 Background

Although there have been significant prior studies in NLG and benchmarks for Bengali (Bhattacharjee et al., 2022; Ekram et al., 2022; Raihan et al., 2025a), the number of code generation studies using LLM is quite negligible. The only substantial study we could find is a family of finetuned models named *TigerCoder*, which was evaluated for its machine translation capabilities (Raihan et al., 2025b). These findings underscore the need for further exploration using alternative techniques to improve code generation capabilities.

Test-Driven Development (TDD) has been a widely researched methodology in the agile soft-

[1] https://github.com/sajedjalil/BLP25-Task-2/
[2] https://chatgpt.com/

Proceedings of the Second Workshop on Bangla Language Processing (BLP-2025), pages 571–576
December 23, 2025

ID	**Instruction** (Bengali)	**English** (Translated for reader's convenience)	**Test List**
1	একটি পাইথন ফাংশন লিখুন nth বেল নম্বর খুঁজে পেতে। Example: bell_Number(n)	Write a python function to find nth Bell number. Example: bell_Number(n)	assert bell_Number(2)==2 … … other tests … …
2	একটি জটিল সংখ্যার দৈর্ঘ্য পেতে একটি ফাংশন লিখুন। Example: len_complex(n, n2)	Write a function to find the magnitude of a complex number. Example: len_complex(n, n2)	assert len_complex(3,4)==5.0 … … other tests … …

Figure 1: Example of dataset rows used in our study (English instruction is added here for readers' convenience.)

ware engineering domain (Shull et al., 2010; Rafique and Mišić, 2012). It is the practice of writing unit tests before starting implementation to ensure software verification. This methodology has been proven to reduce code defects (Williams et al., 2003). To the best of our knowledge, no other prior studies have explored the effects of TDD in code generation with Bengali prompts.

Code Interpreter (CI) can act as an external tool to help LLM improve itself as a coding agent (Wang et al., 2024). Humans interact with LLM multiple times if the desired output is not reached (Lin et al., 2025). This inspired us to utilize CI to enhance accuracy and minimize compilation errors in our study. Additionally, we employed a combined approach that incorporates TDD and CI to improve accuracy further and reduce compilation errors.

3 Task Dataset

The primary aim of this task was to generate Python code from Bengali instructions using LLM (Raihan et al., 2025a,c,b). All of our code and experimental results are publicly available on GitHub. [3]

In Figure 1, a sample of the dataset is shown. The test cases evaluate the generated code. Only one test case was publicly available during the competition. The rest were hidden and could only be accessed after the submission phase had ended.

Model family	Used variants
Meta Llama 3.2	3B, 11B, 90B
Meta Llama 4	Scout 17B, Maverick 17B
OpenAI gpt-oss	20B, 120B

Table 1: Distribution of LLM models and variants in our experiment.

4 Experiments

Our LLM responses were generated with the AWS Bedrock platform [4]. Therefore, our selection of various models was dependent upon the availability in Bedrock. Specifically, we experimented with the following models in Table 1.

Our initiative focused on improving the accuracy of generalized LLM code generation without fine-tuning. To achieve this, we have experimented with the following five approaches -

4.1 Vanilla (Baseline) Model

To establish baseline accuracy with the Bengali instruction, we conducted this experiment with plain (vanilla) LLM API to determine how different LLMs perform.

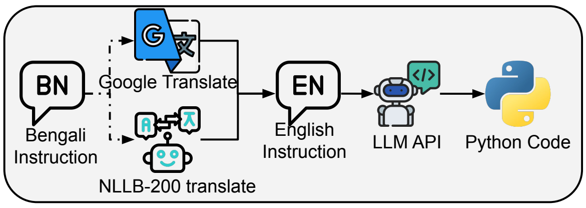

Figure 2: Two variants of Bengali to English machine translation.

4.2 Bengali to English Machine Translation

Since the primary language for most LLMs is English, our initial intuition was to translate the Bengali instructions into English. For this experiment, we have used two different translators - Google Translate [5] & NLLB-200 (Costa-Jussà et al., 2022). The overall workflow of this approach is displayed in Figure 2.

[3] https://github.com/sajedjalil/BLP25-Task-2/

[4] https://aws.amazon.com/bedrock/

[5] https://translate.google.com/

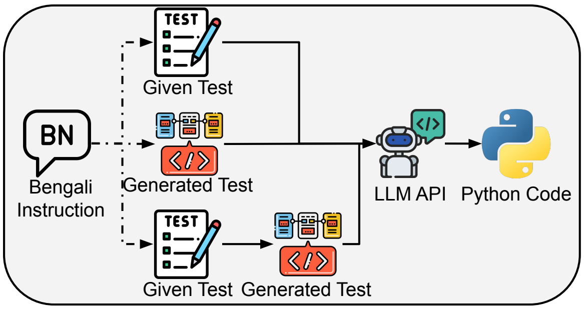

Figure 3: Variants of Test-Driven Development (TDD) approaches in our experiments.

4.3 Test-Driven Development (TDD)

We experimented with three different variations of TDD in this experiment. The detailed diagram is shown in Figure 3.

1. **Generated Tests** - We started with an API call to an LLM to generate up to five test cases from the given prompt. We then input these test cases, along with the given prompt, to generate our final response.
2. **Given Test** - We injected only the publicly available assert statement (test case) from the dataset into the LLM prompt during code generation.
3. **Combined** - This approach combined the above two methods. Here, we used the given test case from the dataset, along with five more LLM-generated test cases. And then, all of these test cases were used in LLM for response generation.

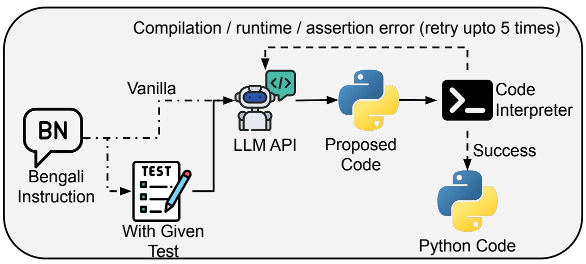

Figure 4: Code Interpreter with Test-Driven Development (TDD) approach.

4.4 Code Interpreter (CI)

We drew inspiration for this method from how developers interact with LLMs in real life. Developers generate code from LLM and then test the code in their respective IDE or environment. If any problem is encountered, they continue the chat and share error messages with the LLM until the desired output is achieved.

We utilized AWS Code Interpreter as a simulated Python environment [6]. Given a Python code, it compiled and executed it. For any errors, a detailed error message was obtained. We also set a retry limit of five to fix the generated Python code that did not compile. The error message received from the earlier execution was used as additional input for subsequent code generation. A workflow model is displayed in Figure 4 with the vanilla path.

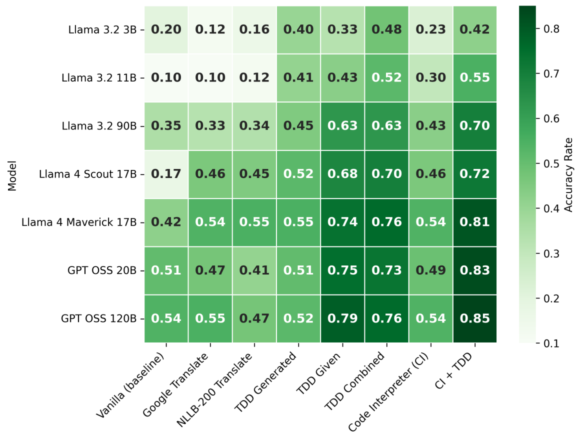

Figure 5: Overall accuracy heatmap of models in different approaches.

4.5 CI+TDD

We combined the TDD approach with the dataset provided single test case with the CI. This test case was also executed in the interpreter to verify its success. The mechanism is demonstrated in Figure 4 with the given test path.

5 Results & Analysis

The results of our experiments are provided in Table 2 and Table 3. We group the data by model family and model parameters. The best outcome for each model is represented by bold text.

RQ1: ***How far can performance improve without fine-tuning or external data augmentation?***

In our investigation, we obtained several interesting findings. Figure 5 demonstrates the overall accuracy score on the test phase. It is distinctly evident from the heatmap that vanilla (baseline) accuracy can be improved significantly with TDD and CI. Except for the Llama 4 models, machine translation from Bengali to English did not provide

[6] https://docs.aws.amazon.com/bedrock/latest/userguide/agents-code-interpretation.html

Model	Vanilla	Translated		Test-Driven Development			Code Interpreter	
		Google	NLLB	Generated	Given	Combined	Vanilla	Given Test
Llama 3.2 models								
3B	19.6	12.4	16.0	39.6	33.4	**48.2**	22.6	42.2
11B	9.8	9.6	12.4	40.6	42.8	51.8	30.4	**54.8**
90B	35.0	32.8	34.2	44.8	62.8	63.4	42.6	**69.6**
Llama 4 models								
Scout	16.8	45.8	45.4	51.6	67.6	69.6	45.8	**72.0**
Maverick	42.0	54.0	54.6	54.6	74.4	76.4	54.4	**80.6**
GPT-OSS models								
20B	51.0	47.0	41.4	50.8	75.4	72.6	48.6	**82.8**
120B	54.4	54.6	46.8	52.2	79.0	75.6	54.0	**85.0**

Table 2: Overall accuracy (%) on varying model family and parameter size over different approaches.

Model	Vanilla	Translated		Test-Driven Development			Code Interpreter	
		Google	NLLB	Generated	Given	Combined	Vanilla	Given Test
Llama 3.2 models								
3B	21.4	61.8	38.4	8.4	7.6	5.8	**0.2**	0.8
11B	67.8	71.2	61.0	5.2	0.2	2.0	**0.0**	0.2
90B	8.8	15.4	9.4	**0.2**	**0.2**	0.4	0.4	**0.2**
Llama 4 models								
Scout	66.4	1.2	1.4	3.6	0.8	0.8	**0.2**	2.6
Maverick	19.2	0.2	0.2	1.2	**0.0**	0.2	0.2	**0.0**
GPT-OSS models								
20B	1.0	1.8	1.8	1.8	1.2	1.2	**0.2**	**0.2**
120B	0.2	0.4	0.2	0.4	0.8	1.2	**0.0**	0.2

Table 3: Overall compilation errors occurrence (%) on varying model family and parameter size.

significant improvement. Instead, it harmed the overall accuracy compared to the non-translation approach.

On the other hand, we observed an impressive increase in accuracy compared to the baseline in Figure 6. The CI+TDD approach improved the accuracy across all models by **+57%** to **+450%**. The TDD approach improves the baseline by **+47%** to **+420%**. Bengali to English machine translation has a change factor from **-20%** to **+171%**.

> **Compared to baseline (54%), overall accuracy can be improved up to 85% using our proposed techniques.**

RQ2: ***Can smaller models approach larger model performance?***

In the Llama 3.2 model family, using the TDD 3B variant (48%) exceeds the baseline accuracy of the 90B variant (35%). Comparing the best outcome for each variant, we observed that 3B could reach **67%** of the performance of 90B and **87%** of the performance of 11B.

For the Llama 4 model family, Scout can exceed the Maverick baseline by 71% using CI+TDD. When comparing best outcomes, Scout can achieve up to **89%** of Maverick's performance.

Lastly, in the gpt-oss variants, the 20B variant using CI+TDD surpasses the 120B baseline by 54%. In best-case scenarios for both, 20B can reach up to **98%** of the performance of 120B. Our results further confirm the claims made by another prior study (Belcak et al., 2025).

> **In the same model family, the smallest model can attain up to 98% accuracy of the largest model.**

RQ3: ***What approach is most effective in improving vanilla (baseline) LLM accuracy?***

As Figure 6 indicates that the best outcome is

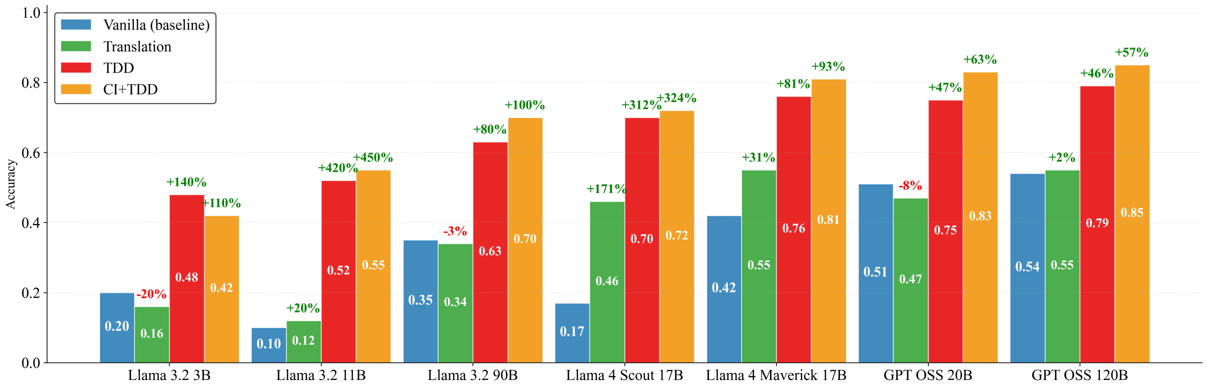

Figure 6: Model accuracy comparisons of baseline vs. our approaches (with increase/decrease in percentage).

always from CI+TDD except for the Llama 3.2 3B model. It should be noted that both TDD and CI+TDD performed significantly better than baseline in all models.

Combination of Test-Driven Development (TDD) and Code Interpreter (CI) yields the largest jump in accuracy.

Figure 7: Compilation error rate heatmap of models in different approaches.

RQ4: ***To what extent do these approaches reduce compilation errors?***

In terms of total compilation errors in the generated code, a similar trend is visible as the accuracy rate. Figure 7 demonstrates both TDD and CI approaches have nearly eliminated all compilation errors with rates approaching **0%**. Translation helped reduce compilation errors only in the Scout model families. An interesting trend is observed in the gpt-oss model family, whose baseline compilation errors are nearly zero, suggesting it may contain inherent mechanisms to address compilation issues.

Both TDD and CI reduce compilation errors, whereas Bengali to English machine translation increases the error count in most cases.

Conclusion

This study successfully introduced a novel approach that combines **Test-Driven Development (TDD) and Code Interpreter (CI)** to improve code generation accuracy for Bengali prompts utilizing open-weight LLMs. Our findings demonstrate that this strategy yields significant improvements without requiring resource-intensive fine-tuning or the use of external data for augmentation. The **CI+TDD methodology** was the most effective, increasing overall baseline accuracy by up to 450% and virtually eliminating compilation errors across all models tested. Furthermore, our research suggests that using these strategies, even the smallest models in the same family can achieve up to **98% accuracy** when compared to the largest models of the same family. We strongly believe the exact mechanism can be applied to other underrepresented languages, similar to Bengali, and increase access to high-performing code generation tools in resource-constrained emerging markets.

6 Limitations

Our study is limited in context, as we only checked a subset of open-weight models available on the AWS Bedrock API. This restricted us from using several other popular models not available on that platform, such as Qwen3 and Gemma3. Moreover, we did not explore how our approach would perform in larger and complex coding tasks, as opposed to the single method-based problems provided in the shared-task dataset.

Acknowledgments

We greatly appreciate Apollo Applications Group [7] for granting us access to the AWS resources necessary to perform this study. And special thanks to Marium Binte Ibrahim Ema and Jeffrey Dershewitz for their reviews.

References

Mehdi Ali, Michael Fromm, Klaudia Thellmann, Richard Rutmann, Max Lübbering, Johannes Leveling, Katrin Klug, Jan Ebert, Niclas Doll, Jasper Buschhoff, and 1 others. 2024. Tokenizer choice for llm training: Negligible or crucial? In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3907–3924.

Peter Belcak, Greg Heinrich, Shizhe Diao, Yonggan Fu, Xin Dong, Saurav Muralidharan, Yingyan Celine Lin, and Pavlo Molchanov. 2025. Small language models are the future of agentic ai. *arXiv preprint arXiv:2506.02153*.

Abhik Bhattacharjee, Tahmid Hasan, Wasi Uddin Ahmad, and Rifat Shahriyar. 2022. BanglaNLG and BanglaT5: Benchmarks and resources for evaluating low-resource natural language generation in bangla. *arXiv preprint arXiv:2205.11081*.

Marta R Costa-Jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, and 1 others. 2022. No language left behind: Scaling human-centered machine translation. *arXiv preprint arXiv:2207.04672*.

Syed Mohammed Sartaj Ekram, Adham Arik Rahman, Md Sajid Altaf, Mohammed Saidul Islam, Mehrab Mustafy Rahman, Md Mezbaur Rahman, Md Azam Hossain, and Abu Raihan Mostofa Kamal. 2022. Banglarqa: A benchmark dataset for under-resourced bangla language reading comprehension-based question answering with diverse question-answer types. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2518–2532.

Sajed Jalil. 2025. The transformative influence of llms on software development & developer productivity. In *2025 International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA)*, pages 1–10.

Xinrui Lin, Heyan Huang, Kaihuang Huang, Xin Shu, and John Vines. 2025. Seeking inspiration through human-llm interaction. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–17.

Junho Myung, Nayeon Lee, Yi Zhou, Jiho Jin, Rifki Putri, Dimosthenis Antypas, Hsuvas Borkakoty, Eunsu Kim, Carla Perez-Almendros, Abinew Ali Ayele, and 1 others. 2024. Blend: A benchmark for llms on everyday knowledge in diverse cultures and languages. *Advances in Neural Information Processing Systems*, 37:78104–78146.

Yahya Rafique and Vojislav B Mišić. 2012. The effects of test-driven development on external quality and productivity: A meta-analysis. *IEEE Transactions on Software Engineering*, 39(6):835–856.

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).

Forrest Shull, Grigori Melnik, Burak Turhan, Lucas Layman, Madeline Diep, and Hakan Erdogmus. 2010. What do we know about test-driven development? *IEEE software*, 27(6):16–19.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better LLM agents. In *Forty-first International Conference on Machine Learning*.

Laurie Williams, E Michael Maximilien, and Mladen Vouk. 2003. Test-driven development as a defect-reduction practice. In *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.*, pages 34–45. IEEE.

[7] https://www.apolloapps.ai/