JU_NLP at BLP-2025 Task 2: Leveraging Zero-Shot Prompting for Bangla Natural Language to Python Code Generation

Pritam Pal and Dipankar Das
Jadavpur University, Kolkata, India
{pritampal522, dipankar.dipnil2005}@gmail.com

Abstract

Code synthesis from natural language problem statements has recently gained popularity with the use of large language models (LLMs). Most of the available systems and benchmarks, however, are developed for English or other high-resource languages, and a gap exists for low-resource languages such as Bangla. Addressing the gap, the Bangla Language Processing (BLP) Workshop at AACL-IJCNLP 2025 featured a shared task on Bangla-to-Python code generation. Participants were asked to design systems that consume Bangla problem statements and generate executable Python programs. A benchmark data set of training, development, and test splits was provided, and evaluation utilized the Pass@1 metric through hidden test cases. We present here a system we developed, using the state-of-the-art LLMs through a zero-shot prompting setup. We report outcomes on several models, including variants of GPT-4 and Llama-4, and specify their relative strengths and weaknesses. Our best-performing system, based on GPT-4.1, achieved a Pass@1 score of 78.6% over the test dataset. We address the challenges of Bangla code generation, morphological richness, cross-lingual understanding, and functional correctness, and outline the potential for future work in multilingual program synthesis.

1 Introduction

The intersection of program synthesis and natural language processing has witnessed significant progress over recent years, with much of the work being done by large language models (LLMs). Code generation tasks, where a natural language expression of a problem is automatically converted into executable source code, have been explored extensively in English and specific high-resource languages (Lu et al., 2021; Chen et al., 2021; Austin et al., 2021). Very little work has been conducted on this task in low-resource languages, such as South Asian languages like Bangla (Raihan et al., 2025b), despite Bangla being one of the world's 10 most widely spoken languages and the national language in Bangladesh and second most widely spoken language in the Indian subcontinent.

To bridge this gap, a Shared Task on Code Generation for Bangla was organized as a task at the Bangla Language Processing (BLP) workshop at AACL-IJCNLP 2025 (Raihan et al., 2025c). The shared task entailed developing automatic systems to generate Python source code from Bangla problem statements. The organizers released a benchmark dataset with Bangla problem descriptions and corresponding Python solutions. The participants were asked to develop models/pipelines with the capacity to, upon being given a problem statement in Bangla, generate correct and executable Python programs.

This task is particularly challenging for several reasons. First, Bangla is a resource-scarce and morphologically dense language; thus, there are limited large-scale annotated data resources available for training. Second, we aim to bridge cross-lingual semantic understanding (natural language in Bangla) and code generation at its formal level (in Python), which can result in translation errors leading to syntactic or logical execution failures. Third, creating measures that capture more than surface-level textual similarity and move towards the functional correctness of code generation presents an additional level of challenge.

The shared task provides to the community a standardized dataset, evaluation framework, and collection of baseline results to promote orderly progress in this new field. Beyond benchmarking, it is desirable to stimulate the development of multilingual and cross-lingual code generation models and to support broader efforts to make programming more inclusive for speakers of Bangla and other learners and programmers.

Proceedings of the Second Workshop on Bangla Language Processing (BLP-2025), pages 582–586

2 Dataset

The datasets provided by the BLP-2025 Task 2 shared task organizers were used to develop the code generation framework. The organizers first provided a trial dataset to help understand the task and the input/output formats. Next, a development dataset (Raihan et al., 2025a) was provided by the organizers for the code generation framework, and a test dataset to evaluate its performance. The trial dataset comprises 74 problem statements in the Bengali language, each accompanied by corresponding Python source code and three executable test cases.

In the development dataset, there are a total of 400 problem statements in Bengali, with three test cases provided for each statement. The test dataset comprises 500 distinct problem statements in Bengali, which are entirely separate from those in the development dataset. This dataset is used to assess the performance of the code generation framework. To increase the challenge, the organizers only provided one test case for each problem statement in the test dataset, as opposed to the three provided in the development dataset. An example of the data from both the development and test datasets is shown in Figure 1.

Ex - 1

Problem Statement: প্রথম `n` টি প্রাকৃতিক সংখ্যার ঘনক্ষেত্রের যোগফল খুঁজে বের করার জন্য একটি পাইথন ফাংশন লিখুন। (**T:** Write a Python function to find the sum of the cubes of the first n natural numbers.)

```
Exammple:
sum_Of_Series(n)
```

Test Cases: `['assert sum_Of_Series(5) == 225', 'assert sum_Of_Series(2) == 9', 'assert sum_Of_Series(3) == 36']`

Ex - 2

Problem Statement: একটি প্রদত্ত ম্যাট্রিক্সকে তার সারিগুলির যোগফল অনুযায়ী ক্রমবর্ধমান ক্রমে সাজানোর জন্য একটি ফাংশন লিখুন। (**T:** Write a function to sort a given matrix in ascending order according to the sum of its rows.)

```
Example:
def sort_matrix(M):
    # your code
    return M
```

Test Case: `['assert sort_matrix([[1, 2, 3], [2, 4, 5], [1, 1, 1]])==[[1, 1, 1], [1, 2, 3], [2, 4, 5]]']`

Figure 1: Example of development data (Ex-1) and test data (Ex-2). The development data contains three test cases, while the test data contains only one test case.

3 Methodology

This section provides a brief overview of the overall methodology for code generation in the Bengali language. To develop the framework, we utilized state-of-the-art LLMs with a zero-shot setting.

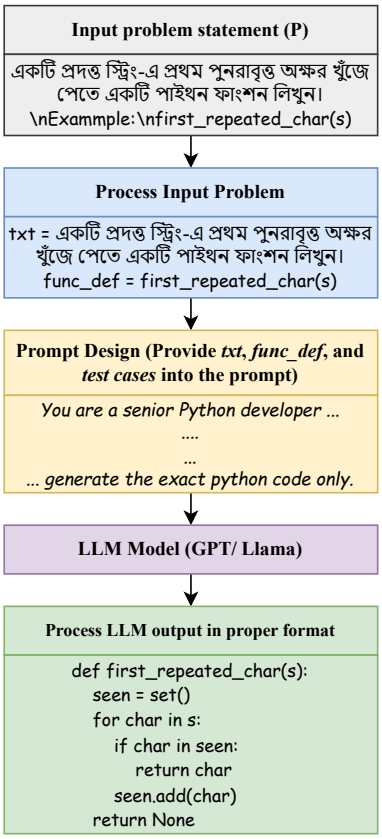

Figure 2: Overall framework of our code generation system. The pipeline begins with processing the Bangla problem statement and function definition, followed by carefully designed prompts. These prompts are then passed to an LLM, and the generated output is post-processed into valid Python code.

Task Definition: Given a problem statement P in the Bengali language, and test cases $T = \{t_1, t_2, ..., t_k\}$, where k is the number of test cases. Our primary objective is to develop a pipeline or framework that can accurately understand the Bengali problem statement and generate Python code that satisfies the test cases T.

Framework Description: The overall flow diagram of the framework is illustrated in Figure 2. Given its state-of-the-art performance across various tasks, such as machine translation, text summarization, sentiment analysis, and many others, we utilized LLMs, including `GPT-4.x` (OpenAI et al., 2024) and `Llama-4` (Touvron et al., 2023), as the core of the framework.

In Figure 2, the first step involves processing the input data provided by the organizers, which consists of two components. The first component is the problem statement, written in Bengali, followed

by the text "\nExample\n", and then the function definition, which includes the function name and parameter list. This process is detailed in the 'Input Problem Statement' section of Figure 2.

We begin by processing the input data and storing it in two separate variables: one for the actual problem statement and the other for the function definition, along with its corresponding parameter list. This will facilitate further processing and prompt design.

The next phase involves designing the prompt. Since LLMs are trained on a vast amount of data and are capable of performing many complex NLP tasks, we employed the zero-shot prompting strategy (Brown et al., 2020; Kojima et al., 2022), where no proper example of the input-output structure is provided in the prompt. Each prompt was designed to include the problem statement (in the original Bengali text), the function definition, and parameter list, allowing for the development of the desired function. Along with the problem statement and function definition, the test cases were provided in the prompt so that the LLM could evaluate them to check whether it correctly generated the desired code. The prompt that was provided to the LLM models to generate the Python source code is provided in Figure 3.

Each LLM model was accessed with its corresponding API keys. During the generation of Python codes, the `temperature` value, `top_p` value, and maximum token limit were set to 0.2, 1, and 1024, respectively, across all LLM models.

Next, the generated Python code outputs were further processed to remove common code-block markdown markers (eg, "``` Python" or "```"). An example of generated Python code from a given problem statement is provided in Figure 2.

```
You are a senior Python developer. Your task is to
write a single, well-structured Python function that
solves the user's problem.

Your output must contain only the code, with no
conversational text. Only generate the proper
executable code.

It is to be noted that the problem statement is
given in the Bengali language. You can translate the
problem statement into English before writing the
code.

Problem: {txt}.

The function definition, return type, and structure
of code should be as follows: {func_def}

There are test cases given in the program. Note
that the generated code should pass the given test
cases.

Test Cases: {test_cases}

Note that there might be other test cases also,
along with the given test cases. Your solution
should also be robust enough to handle other
hidden test cases.

Don't print the test cases. Only generate the
exact Python code.
```

Figure 3: Prompt that was provided to LLMs for generating Python source code from Bengali problem statements. In this prompt, we instructed LLMs to translate the Bengali problem statement to English.

4 Experiment and Result

4.1 Experimental Setup

All experiments were conducted in the Google Colaboratory environment using a non-GPU virtual system with 12.7 GB of RAM. The GPT models were accessed via OpenAI's official website [1], while the Llama-4 model was obtained from a third-party provider, `'Together.AI'` [2], utilizing the appropriate API keys. The experiments were set up in two configurations: one in which the LLM models were instructed to translate the provided Bengali problem statement into English (Figure 3), and another in which the problem statements were left in their original Bengali form [3].

In order to evaluate the framework's performance, the Pass@1 metric was used, which is the percentage of hidden test cases that the generated Python code passes. The experiments were performed using different versions of GPT models, including `GPT-4-mini`, `GPT-4.1-mini`, `GPT-4.1`, and the `Llama-4`[4] model. To ensure a fair comparison, all the LLM models in both experimental setups were executed with the same `temperature`, `top_p`, and maximum token value as described in Section 3.

4.2 Result

The results for the test dataset are presented in Table 1 for both the translated and original Bengali

[1] https://openai.com/

[2] https://www.together.ai/

[3] Prompt for this experimental setup is provided in Appendix A

[4] Exact name is: `Llama-4-Maverick-17B-128E-Instruct-FP8`

LLM Model	Pass@1 (%)	Is translated to English
GPT 4o-mini	72.0	✓
GPT 4.1-mini	76.0	✓
GPT-4.1	**76.6**	✓
Llama-4	76.4	✓
GPT 4o-mini	65.6	✗
GPT-4.1-mini	74.0	✗
GPT-4.1	**78.6**	✗
Llama-4	75.6	✗

Table 1: Performance of different LLMs on the shared task test dataset, measured using the Pass@1 metric. The result highlighted with **Blue** colour represents the best performance when translating the Bengali problem statement into English (Official shared task submission). The result highlighted in **Green** colour represents the best performance when using original Bengali problem statements.

problem statement forms. From the table, it is evident that the `GPT-4.1` model performs best across both schemes, achieving Pass@1 scores of 76.6% with the translated problem statement and 78.6% with the original Bengali problem statement. The `Llama-4` model follows closely behind, obtaining a Pass@1 score of 76.4% for the translated form and 78.6% for the original form.

Notably, the `GPT-4.1` model demonstrates superior performance with the original Bengali problem statement compared to the translated version. This indicates a strong understanding of the Bengali language for code generation by the `GPT-4.1` model. In contrast, other models such as `GPT-4o-mini`, `GPT-4.1-mini`, and `Llama-4` performed better with the translated problem statement and showed decreased performance when using the original Bengali problem statement.

5 Conclusion and Future Work

This paper presents an LLM-based framework for Python code generation from Bengali problem statements. We used a zero-shot prompting strategy with three variants of GPT models and the Llama 4 model across two experimental setups: one translating the Bengali problem to English and another keeping the Bengali problem statement in its original form. All experiments were conducted on the development and testing datasets provided by the BLP Task 2 organizers. Our experimental results demonstrate that the GPT-4.1 model-based framework outperforms all other frameworks in both experimental setups, particularly when using the Bengali original problem statements, which achieved the best result with a 78.6% pass@1 score.

Future directions would include experimenting with other prompting strategies, such as few-shot or chain-of-thought prompting, and making a comparative analysis between different prompting strategies. Additionally, we will also experiment with other LLM models, such as DeepSeek (DeepSeek-AI et al., 2025) or Gemini (Team et al., 2025), in our future work.

Limitations

Our proposed work does have some potential limitations. First, we only experimented with a zero-shot prompting strategy and did not explore other prompting techniques, such as few-shot learning or chain-of-thought prompting. The performance of our approach may be enhanced by using these advanced strategies.

Second, our experiments were limited to only the GPT and Llama-4 models. We chose these models due to their relatively strong performance across various NLP tasks. However, we have not explored other LLMs, such as DeepSeek, Qwen, or Claude. Financial constraints have prevented us from doing so, as each model incurs costs for API calls.

Third, we did not conduct any explicit training or fine-tuning due to resource limitations. The performance could potentially improve if we fine-tune an LLM model, especially for the code generation task. We plan to attempt fine-tuning an LLM model for code generation in our future work.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *Preprint*, arXiv:2108.07732.

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. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 15 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, and 3 others. 2021. Codexglue: A machine learning benchmark dataset for code understanding and generation. *Preprint*, arXiv:2102.04664.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 9 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

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).

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy Lillicrap, Angeliki Lazaridou, and 5 others. 2025. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

A Appendix A: Prompt where no translation was performed

```
You are a senior Python developer. Your task is to
write a single, well-structured Python function that
solves the user's problem.

Your output must contain only the code, with no
conversational text. Only generate the proper
executable code.

It is to be noted that the problem statement is
given in the Bengali language.

Problem: {txt}.

The function definition, return type, and structure
of code should be as follows: {func_def}

There are test cases given in the program. Note
that the generated code should pass the given test
cases.

Test Cases: {test_cases}

Note that there might be other test cases also,
along with the given test cases. Your solution
should also be robust enough to handle other
hidden test cases.

Don't print the test cases. Only generate the
exact Python code.
```

Figure A.1: Prompt to Bengali-to-Python code generation where the problem statement was kept in its original Bengali form. No translation was conducted.