PyBangla at BLP-2025 Task 2: Enhancing Bangla-to-Python Code Generation with Iterative Self-Correction and Multilingual Agents

Jahidul Islam, Md Ataullha, Saiful Azad
Department of Computer Science and Engineering
Green University of Bangladesh, Dhaka, Bangladesh
jahidul.cse.gub@gmail.com ataullha00@gmail.com saiful@cse.green.edu.bd

Abstract

LLMs excel at code generation from English prompts, but this progress has not extended to low-resource languages. This paper addresses the challenge of Bangla-to-Python code generation by introducing BanglaCodeAct, an agent-based framework that leverages multi-agent prompting and iterative self-correction. Unlike prior approaches that rely on task-specific fine-tuning, BanglaCodeAct employs an open-source multilingual LLM within a Thought–Code–Observation loop, enabling the system to dynamically generate, test, and refine code from Bangla instructions. We benchmark several prominent small-parameter open-source LLMs and evaluate their effectiveness on the mHumanEval dataset for Bangla NL2Code. Our results show that Qwen3-8B, when deployed with BanglaCodeAct, achieves the best performance, with a pass@1 accuracy of 94.0% on the development set and 71.6% on the blind test set. These findings establish a new benchmark for Bangla-to-Python translation and highlight the potential of agent-based reasoning for reliable code generation in low-resource languages.. Experimental scripts made publicly available at github.com/jahidulzaid/PyBanglaCodeActAgent

1 Introduction

Large Language Models (LLMs) has created a paradigm shift in software engineering, automating complex coding tasks, and democratizing programming for a larger audience (Chen et al., 2021). Natural Language-to-Code (NL-to-Code) generation (Yin et al., 2022), once a distant goal, is now a tangible reality, with systems capable of producing functional code from simple English descriptions. The vast majority of these advances remain linguistically monolithic, centered almost exclusively on English, leaving other languages behind.. This linguistic bias creates a significant accessibility gap for millions of learners worldwide whose primary language is not English. For speakers of low-resource languages like **Bangla**, the seventh most spoken language globally.

To address this critical issue, we introduce the **BanglaCodeAct Agent**, a ReAct agent framework designed for cross-lingual code generation from Bangla instructions into executable Python. Instead of relying on task-specific fine-tuning, our approach leverages the emergent multilingual reasoning capabilities of a general-purpose open-source LLMs within an iterative, self-correcting loop.

We address 3 research questions:

1. **RQ1:** How can an agent-based framework be designed to generate Python code from natural language instructions in Bangla?

2. **RQ2:** Can a general-purpose, multilingual LLMs be effectively prompted to perform cross-lingual code generation in a zero-shot setting, without the need for task-specific datasets?

3. **RQ3:** How does incorporating an iterative *Thought-Code-Observation* loop within a robust execution environment affect the reliability and correctness of the generated code?

2 Related Work

This research is positioned at the confluence of several rapidly advancing domains: automated code generation, the development of specialized large language models for programming, and the specific challenges within Natural Language Processing (NLP) for low-resource languages like Bangla. Our work synthesizes insights from these areas to address a novel problem: agent-driven,

Proceedings of the Second Workshop on Bangla Language Processing (BLP-2025), pages 566–570

cross-lingual code generation from a low-resource language.

The introduction of the Transformer architecture (Vaswani et al., 2017) created major progress in this field. This led to the development of Large Language Models (LLMs) trained on vast web-scale corpora. Models like OpenAI's Codex, the engine behind GitHub Copilot (Chen et al., 2021), and DeepMind's AlphaCode (Li et al., 2022a), which achieved competitive performance in programming contests. However, a significant limitation of this era has been a reliance on English-centric data and evaluation benchmarks.

Building on the success of general-purpose LLMs, a new wave of models has been specifically trained or fine-tuned for programming tasks. Notable examples include CodeLlama (Roziere et al., 2023); StarCoder (Li et al., 2023); and DeepSeek Coder (Guo et al., 2024).

To reduce hallucination and improve factual grounding, Retrieval-Augmented Generation (RAG) retrieves relevant documents from an external knowledge base and supplies them as context to the LLMs(Lewis et al.). Corrective RAG (CRAG) introduces a lightweight retrieval evaluator to augment retrieved documents, improving the correctness of the generation process (Yan et al., 2024).

Bangla NLP faces persistent gaps that make NL2Code especially challenging. First, **data scarcity**: large-scale parallel corpora of Bangla programming instructions and code are virtually absent (Zhong et al., 2024; Raihan et al., 2025a). Second, **morphological complexity**: Bangla's rich inflectional system makes natural language instructions harder to parse into precise logical forms compared to English (Bhattacharjee et al., 2023). Prior LLM-based approaches, trained or fine-tuned primarily on English or multilingual data, often fail to capture these nuances, resulting in low accuracy and unstable performance in Bangla NL2Code tasks (Chen et al., 2021; Li et al., 2022b).

Our work addresses these gaps by introducing **BanglaCodeAct**, which directly leverages the multilingual reasoning abilities of general-purpose LLMs in a self-correcting loop, without requiring costly Bangla-specific fine-tuning or large annotated datasets.

3 Dataset and Evaluation Metrics

The task involves translating Bangla natural language programming instructions into Python code, ensuring functional correctness by passing associated test cases. This setup mirrors typical NL2Code challenges but places a specific emphasis on low-resource language understanding and algorithmic reasoning in Bangla (Raihan et al., 2025c). To evaluate this translation process, we employ the **mHumanEval dataset** (Raihan et al., 2025a), which is tailored for Bangla-to-Python code generation. The dataset consists of natural language programming problems in Bangla, each paired with a corresponding Python implementation and unit test cases that serve as an objective correctness signal. It covers a wide range of fundamental programming concepts, including algorithmic reasoning, control structures, data manipulation, and function design. The sample structure of the dataset is presented in Table 1.

3.1 Evaluation Metric

The primary metric for our evaluation is **pass@1** on the HumanEval benchmark (Raihan et al., 2025a). A generated code snippet is considered a "pass" if it executes without error and satisfies all provided assertions in the 'test_list' for that problem.

$$\text{pass@}k := E_{\text{problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right]$$

4 Methodology

In this work we introduce an agent framework, **BanglaCodeAct Agent**, for cross-lingual code generation. The primary objective is to translate natural language programming instructions articulated in a low-resource language, **Bangla (Bengali)**, into executable Python code. The methodology hinges on a powerful multilingual Large Language Models (LLMs) integrated into an iterative reasoning and self-correction loop, enabling it to bridge the semantic gap between Bangla prose and Python's formal syntax.

4.1 Models and Baselines

We compare the performance of our proposed **BanglaCodeAct Agent** against several baselines to evaluate the contribution of its components and to situate its performance relative to other

ID	Instruction (Bengali)	Test Cases
1	একটি ফাংশন লিখুন যা পরীক্ষা করবে প্রদত্ত স্ট্রিং প্যালিনড্রোম কিনা। খালি স্ট্রিংক প্যালিনড্রোম হিসেবে গণ্য হবে। Example: `is_palindrome(s)`	assert is_palindrome("TENET") == True assert is_palindrome("Bangla") == False assert is_palindrome(" ") == True
2	একটি ফাংশন লিখুন যা একটি স্ট্রিং-এর মধ্যে থাকা শব্দগুলোকে উল্টো করে সাজাবে। Example: `reverse_words(string)`	assert reverse_words("hello")=="hello" assert reverse_words(" a b ") == "b a" assert reverse_words("hello world") =="world hello"
3	একটি পাইথন ফাংশন লিখুন যা দিয়ে দুইটি পূর্ণসংখ্যার বিপরীত চিহ্ন আছে কিনা তা পরীক্ষা করা যায়। Example: opposite_Signs(n1, n2)	assert opposite_Signs(1,-2) == True assert opposite_Signs(3,2) == False assert opposite_Signs(-10,-10) == False

Table 1: The dataset for Shared Task 2 (Code Generation) includes Bengali programming instructions, the corresponding Python code implementations, and test cases designed for validation.

approaches. First, **Zero-Shot Prompting** serves as a direct baseline where the model is given only the system prompt and the user task (Bangla instruction plus test cases) and is asked to generate the solution in a single turn. This approach achieves varying results across models: Qwen/Qwen3-8B obtains 36%, Qwen-Coder-7B reaches 51%, TigerLLM-1B-it (Raihan et al., 2025b) it achieves 11%, and Llama-3.1-8B performs the best at 77%. Next, **Few-Shot Prompting** provides the model with a small number of solved examples in the prompt to help it generalize to new problems. Performance here also varies, with Qwen3-8B achieving 46%, Qwen2.5-Coder-7B reaching 51%, and Llama-3.1-8B again performing strongly at 77%. DeepSeek-Coder-V2-Lite shows competitive results with a pass@1 of 73.0%. The **Self-Consistency** method leverages Qwen/Qwen3-8B to generate multiple independent solutions for the same problem and selects the final answer through majority voting, without using any iterative feedback loop. Finally, our full proposed framework, the **BanglaCodeAct Agent**, based on Qwen/Qwen3-8B, significantly outperforms these baselines with a 94% success rate. This agent employs an iterative *Thought-Code-Observation* loop, allowing it to self-correct based on execution feedback until all test cases are satisfied.

The experiments were executed with inference controlled by the hyperparameters presented in Table 2.

The agent's core is the **Qwen/Qwen3-8B** model, a multilingual LLM capable of zero-shot Bangla-to-logic translation and reasoning. To enable efficient multi-turn reasoning, we deploy it with the **vLLM inference engine**, leveraging **tensor parallelism** and **prefix caching** for reduced latency and high throughput (Kwon et al., 2023).

Parameters	Value
Max tokens	8192
Temperature	0.7
Top-p	0.9
Best-of	1
Repetition penalty	1.05 (CoT)
Decoding	Self-consistency ($n = 5$)
Num paths	16 / 5 (SC)
Seed	42
Timeout	5 Seconds
Retries	25

Table 2: Inference hyperparameters. These decoding and sampling parameters control output length, diversity, reproducibility, and error handling.

4.2 Cross-Lingual BanglaCodeAct Agent Framework

We employ the Code Acting (`CodeAct`) paradigm to structure the agent's problem-solving process. This approach transforms code generation from a single-shot task into a dynamic, multi-step dialogue between the agent and a code interpreter. The agent operates on a *Thought-Code-Observation* cycle (as illustrated in Fig. 1):

1. **Thought:** The agent generates an internal monologue, outlining its understanding and plan for the task in `<thought>`, showcasing reasoning in Bangla before code generation.

2. **Code Generation:** The agent produces Python code based on the plan, enclosed in `<code>` with test assertions for immediate self-verification.

3. **Execution and Feedback:** The code runs in a sandboxed **PythonREPL** with a timeout. Errors, like `TypeError`, provide feedback for **iterative self-correction**, refining the solution until valid or a max iteration is reached, with the result in `<answer>`.

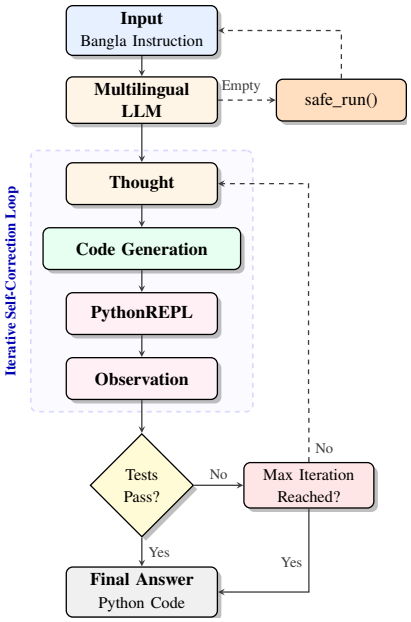

Figure 1: Thought-Code-Observation Cycle in the BanglaCodeAct Agent Framework. This diagram illustrates the iterative process of generating code, executing it, providing feedback, and refining the solution based on self-correction, facilitating cross-lingual code generation in Bangla.

To enhance reliability, we implement a **retry handler** (`safe_run`) that re-initiates the reasoning process if the agent produces an invalid or empty response. The retry mechanism permits a user-defined number of task attempts, improving success rates by reducing sporadic failures.

Instruction (Bengali)	Test Cases
স্ট্রিং থেকে প্রদত্ত অক্ষরের প্রথম এবং শেষ উপসর্গ মুছে ফেলুন। Example: `remove_Occ(s, ch)`	remove_Occ("hello","l") == "heo" remove_Occ("banana","a") == "bann" remove_Occ("abc","x") == "abc"
একটি প্রদত্ত ম্যাট্রিক্সকে তার সারিগুলির যোগফল অনুযায়ী সাজান। Example: `sort_matrix(M)`	sort_matrix([[1,2,3],[2,4,5],[0,1,1]]) == [[0,1,1],[1,2,3],[2,4,5]] sort_matrix([[5,5],[2,2],[3,3]]) == [[2,2],[3,3],[5,5]]

Table 3: Illustrating error recovery in ambiguous and complex cases.

For instance, "স্ট্রিং থেকে প্রদত্ত অক্ষরের প্রথম এবং শেষ উপসর্গ মুছে ফেলুন" (remove the first and last occurrence of a given character from a string). Initial attempts produced incomplete logic (removing only one occurrence). (see Table 3).

5 Results and Analysis

Different models and experiments were conducted during the development phase, which are reported in 4.1. The experiment setup and hyperparameter details are described in table 2.

The 'pass@1' scores for all evaluated methods on the mHumanEval dataset are summarized in Table 4. Our proposed BanglaCodeAct Agent achieves a 'pass@1' score of **94.0%**, significantly outperforming all other methods.

LLM Model	Method	pass@1
Qwen3-8B	**BanglaCodeAct**	**94.0**
Qwen3-8B	Self-Consistency	88.0
Qwen3-8B	Majority Voting	66.0
Qwen3-8B	Few-Shot	46.0
Qwen3-8B	Zero-Shot	36.0
Qwen2.5-Coder-7B	Few-Shot	51.0
Qwen2.5-Coder-7B	Zero-Shot	44.0
Llama-3.1-8B	Zero-Shot	39.0
Llama-3.1-8B	Few-Shot	77.0
DeepSeek-Coder-V2-Lite	**BanglaCodeAct**	**73.8**
DeepSeek-Coder-V2-Lite	Few-Shot	73.0
DeepSeek-Coder-V2-Lite	Zero-Shot	71.4
TigerLLM-1B-it	Zero-Shot	11.0

Table 4: Comparison of pass@1 accuracy (%) for different models and prompting strategies on the mHumanEval dataset. Our proposed **BanglaCodeAct Agent** (Qwen3-8B) achieves the highest score, demonstrating the effectiveness of iterative self-correction.

The results in Table 4, clearly demonstrate the efficacy of our agent-based framework.

The experimental results demonstrate the effectiveness of the proposed **BanglaCodeAct Agent** in leveraging an iterative self-correction mechanism for Bangla-to-Python code generation. With the Qwen3-8B model, the agent achieves a **94.0% pass@1 accuracy**, significantly outperforming Zero-Shot (36.0%), Few-Shot (46.0%), and Majority Voting (66.0%) strategies (Table 4). It ranked 17th on the test set (71.6%) and 8th on the development set (94%).

These results underscore the agent's ability to correct common code generation errors using REPL feedback, which distinguishes it from static prompting approaches. The Qwen3-8B model outperforms specialized models like Qwen2.5-Coder-7B, highlighting the importance of multilingual reasoning over code-specific training. Primary failure cases occur with semantically ambiguous or complex instructions, where the agent may not converge within 10 iterations.

6 Limitations

Despite strong performance, BanglaCodeAct has several limitations. The model's effectiveness is

limited by its size, and experiments with larger LLMs (e.g., 32B parameters or more) were not conducted due to GPU resource constraints. Such models could potentially improve code generation accuracy for more complex tasks.

Additionally, the current evaluation primarily focuses on algorithmic and syntactic correctness. The system's ability to understand semantics and handle ambiguous or context-dependent Bangla instructions remains an open challenge. Moreover, the system relies on high-quality test cases for feedback, which may not always be available in real-world scenarios. The performance could be further limited by the absence of such reliable test cases in practice.

References

Abhishek Bhattacharjee, Shammur Aktar, Md. Saidul Islam Zishan, Sudipta Ghosh, Md. Mahadi Hasan, M. Sohel Rahman, Mohammad Ruhul Alam, Tetsuji Nakagawa, Teruhisa Misu, and Farig Shah. 2023. BanglaBERT: A Denoising Autoencoding based Pre-trained Language Model for Bengali. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2577–2591.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique P. de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, and 1 others. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *CoRR*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. Retrieval-augmented generation for knowledge-intensive nlp tasks.

R Li, LB Allal, Y Zi, N Muennighoff, D Kocetkov, C Mou, M Marone, C Akiki, J Li, J Chim, and 1 others. 2023. Starcoder: May the source be with you! *Transactions on machine learning research*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Dadashi, D O R A Lazic, Petar Veličković, Jiayuan Son, Johanni Botha, and et al. 2022a. Competition-Level Code Generation with AlphaCode. *Science*, 378(6624):1092–1097.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022b. 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. *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).

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Tworkowski, Marie-Anne Lachaux, Thibaut Lavril, Iz Mason, Alexandre Masson, Arthur Mensch, and 1 others. 2023. Code Llama: Open Foundation Models for Code. *arXiv preprint arXiv:2308.12950*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. 2024. Corrective retrieval augmented generation. *Available at SSRN 5267341*.

Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, and 1 others. 2022. Natural language to code generation in interactive data science notebooks. *arXiv preprint arXiv:2212.09248*.

Tianyang Zhong, Zhenyuan Yang, Zhengliang Liu, Ruidong Zhang, Yiheng Liu, Haiyang Sun, Yi Pan, Yiwei Li, Yifan Zhou, Hanqi Jiang, and 1 others. 2024. Opportunities and challenges of large language models for low-resource languages in humanities research. *arXiv preprint arXiv:2412.04497*.