Troopers at BLP-2025 Task 2: Reward-Selective Fine-Tuning based Code Generation Approach for Bangla Prompts

Musa Tur Farazi **Nufayer Jahan Reza**
Department of Computer Science and Engineering (CSE)
Department of Biomedical Engineering (BME)
Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh
{musatur330,nufayerjahanreza8}@gmail.com

Abstract

We present a formally grounded description of a reward-selective fine-tuning (RSFT) pipeline for code generation from Bangla natural-language prompts. The implemented system mines candidate programs via temperature and nucleus sampling, executes candidates in a sandbox and retains programs that pass all unit tests, performs supervised fine-tuning (SFT) on winners using parameter-efficient Low rank adaptation (LoRA) adapters, and augments robustness through fuzzed asserts. We specify the exact objectives and estimators used, provide a Bangla-aware preprocessing recipe, prove simple properties of the sampling budget, and report an ablation showing the effect of inference sample budget K on accuracy. We also include a threat model for safe execution. Our codes are available on GitHub.[1]

1 Introduction

We investigate *reward-selective fine-tuning* (RSFT) of Bangla-to-Python code, a light-weight generate–execute–select–SFT loop that only keeps execution-checked contenders and fine-tunes with maximum likelihood. Unlike RLHF-style policy optimization, RSFT avoids reward modeling and on-policy credit assignment, avoiding instability and engineering overhead. Our system combines stochastic discovery with sandboxed unit tests and fuzzed asserts for safety, and uses LoRA for parameter-efficient adaptation. We then analyze the sample budget K, showing why returns saturate, and see PASS@1/PASS@k ablations as predicted by the theory. Our main contributions in this instance are an industrial RSFT pipeline for Bangla code, fuzzed asserts hardening execution harness, and a slim theory–experiment bridge for the sample budget role.

[1] https://github.com/Musa-Tur-Farazi/BLP-Code-Genenation-Task.git

2 Related Work

Policy-gradient methods and RLHF optimize non-differentiable or preference rewards but are complex and unstable (Ranzato et al., 2016; Paulus et al., 2018; Stiennon et al., 2020; Ouyang et al., 2022).Generate–filter–finetune schemes avoid policy gradients by selecting high-quality samples (RAFT/RSFT; STaR) or by converting preferences into supervised loss (DPO) (Dong et al., 2023; Zelikman et al., 2022; Rafailov et al., 2023). LoRA updates a tiny fraction of weights and pairs naturally with RSFT (Hu et al., 2021). For code models, large-scale supervised/instruction tuning and verification-guided training underpin systems such as Codex, AlphaCode, Code Llama, and CodeRL (Chen et al., 2021; Li et al., 2022; Rozière et al., 2023; Le et al., 2022).We adopt the RSFT recipe with execution correctness as the filter and LoRA for efficient updates.

3 Problem Statement

Let $\mathcal{P}$ be the set of Bangla prompts and $\mathcal{Y}$ the set of syntactically valid Python programs (token sequences). With parameters θ,

$$P_\theta(y \mid p) = \prod_{t=1}^{L} P_\theta\big(y_t \mid p, y_{<t}\big), \tag{1}$$

where $y = (y_1, \ldots, y_L)$.

Each prompt p has a unit-test suite T_p. We use a binary reward

$$r(y; T_p) = \mathbf{1}\{y \text{ passes all tests in } T_p\}, \tag{2}$$

and optionally a fractional reward $r_{\text{frac}}(y; T_p) \in [0, 1]$.

4 Bangla-aware Preprocessing

Unicode normalization. Prompts are normalized to NFC following the Unicode standard to harmonize visually identical but canonically distinct sequences (The Unicode Consortium, 2024).

Proceedings of the Second Workshop on Bangla Language Processing (BLP-2025), pages 561–565
December 23, 2025

Script and punctuation. We preserve Bangla digits and punctuation; ASCII punctuation present in prompts is retained (no transliteration) to avoid corrupting code-like tokens in the target.

Tokenization. Subword tokenization jointly covers Bangla prompts and Python targets using SentencePiece/BPE (Kudo and Richardson, 2018), a choice consistent with recent Bangla language modeling work(Bhattacharjee et al., 2022; Sennrich et al., 2016). We refrain from code-specific token surgerto avoid introducing off-policy artifacts.

5 Mining by Sampling and Sandboxed Execution

Candidates are sampled stochastically and executed under isolation.

Decoding. Let z_t be logits at step t. Temperature $T > 0$ rescales logits to z_t/T. Nucleus (top-p) sampling restricts sampling to the smallest token set with cumulative probability at least p. The miner distribution is $\pi_{\text{old}}(\cdot \mid p)$.

Discovery statistics. Under π_{old},

$$p_{\text{succ}}(p) = \Pr_{y\sim\pi_{\text{old}}(\cdot|p)} \big[r(y; T_p) = 1\big]. \quad (3)$$

With K independent draws, $\mathbb{E}[\text{winners}] = K\, p_{\text{succ}}(p)$ and

$$\Pr(\text{at least one winner}) = 1 - \big(1 - p_{\text{succ}}(p)\big)^K. \quad (4)$$

Threat model and sandboxing. Execution occurs in a restricted environment with no network, including CPU, memory, time limits, constrained file-system and without modules and system calls. Unit tests run solely within this enclave to evaluate $r(y; T_p)$.

6 RSFT Dataset and Supervised Fine-tuning

From the sampled candidates, winners are retained. Let $\mathcal{S}_p$ be the multiset of winners for p. The induced empirical RSFT distribution is

$$Q(y \mid p) = \frac{\pi_{\text{old}}(y \mid p)\, r(y; T_p)}{Z(p)}, \quad Z(p) = \mathbb{E}_{y\sim\pi_{\text{old}}(\cdot|p)}\big[r(y; T_p)\big]. \quad (5)$$

Supervised fine-tuning minimizes the negative log-likelihood on $\mathcal{D}_{\text{rsft}} = \{(p, y)\}$:

$$\mathcal{L}_{\text{MLE}}(\theta) = -\sum_{(p,y)\in\mathcal{D}_{\text{rsft}}} \log P_\theta(y \mid p). \quad (6)$$

Training on samples from Q corresponds to minimizing

$$\mathbb{E}_p[\text{KL}(Q(\cdot \mid p) \,\|\, P_\theta(\cdot \mid p))] .$$

7 Parameter-efficient Fine-tuning

We use LoRA adapters to reduce trainable parameters. For weight matrix $W \in \mathbb{R}^{d_o\times d_i}$,

$$W' = W + \Delta W, \qquad \Delta W = BA, \quad (7)$$

with $A \in \mathbb{R}^{r\times d_i}$, $B \in \mathbb{R}^{d_o\times r}$, $r \ll \min(d_i, d_o)$. Only A and B are updated.

8 Robustness using Fuzzed Asserts

To discourage brittle solutions, some test suites are augmented with perturbed inputs and mutated asserts. If $\mathcal{T}'_p$ denotes the augmented set,

$$r'(y; \mathcal{T}'_p) = \mathbf{1}\{y \text{ passes all tests in } \mathcal{T}'_p\},$$

which tightens the correctness predicate and improves selection precision.

9 Evaluation Metrics

We report PASS@1 with greedy decoding and PASS@k with stochastic sampling following the HumanEval/Codex protocol (Chen et al., 2021). Given n samples with c correct,

$$\widehat{\text{pass@}k} = 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}, \qquad n \ge k, \quad (8)$$

the standard unbiased estimator under sampling without replacement. For uncertainty we use exact Clopper–Pearson or Wilson score intervals (Clopper and Pearson, 1934; Wilson, 1927).

10 Formal Properties of the Sampling Budget K

Let $p = p_{\text{succ}}(p)$ for brevity. The function $f(K) = 1 - (1 - p)^K$ (Eq. 4) is:

- **Monotone** in K for any $p \in (0, 1]$.
- **Concave** in K (diminishing marginal returns), since $f''(K) = -(1 - p)^K \ln^2(1 - p) \le 0$.
- To attain $\Pr(\ge 1 \text{ winner}) \ge 1 - \delta$, it suffices that

$$K \;\ge\; \frac{\ln\delta}{\ln(1-p)} \;\approx\; \frac{1}{p}\ln\frac{1}{\delta} \quad \text{for small } p.$$

These properties explain the empirical concavity with the change in K.

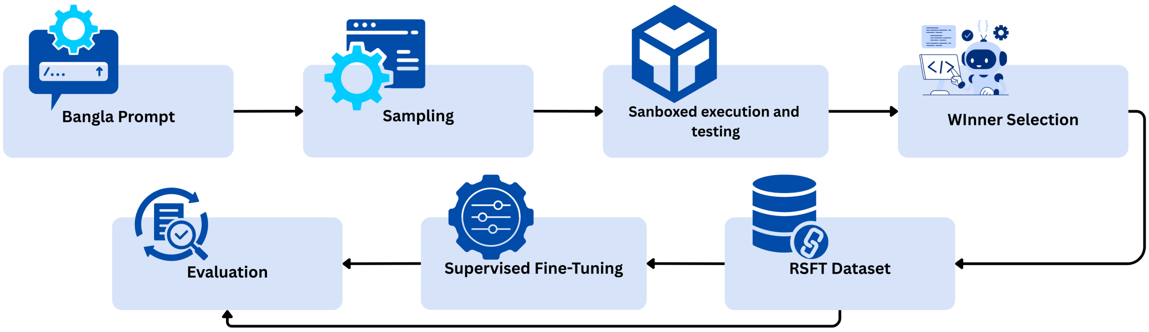

Figure 1: RSFT pipeline: sampling $\rightarrow$ sandboxed execution $\rightarrow$ winner selection $\rightarrow$ SFT with LoRA $\rightarrow$ evaluation.

11 Pseudocode (RSFT Mining & SFT)

Algorithm 1: RSFT Mining

Input: Prompt set $\mathcal{P}$; generator $\pi_{\text{old}}(\cdot \mid p)$; sandboxed executor T_p; optional scorer $s(p, y)$;
Output: Mined supervision pairs D_{rsft}
$D_{\text{rsft}} \leftarrow \varnothing$
foreach $p \in \mathcal{P}$ **do**
  Sample $y^{(1:K)} \sim \pi_{\text{old}}(\cdot \mid p)$
  $S \leftarrow \{\, y^{(j)} \;:\; \textsc{Pass}(T_p(y^{(j)}))\,\}$
  if *s is provided* **then**
    choose $W \subseteq S$ of size m maximizing $s(p, y)$
  else
    choose $W \subseteq S$ of size m
  end
  $D_{\text{rsft}} \leftarrow D_{\text{rsft}} \cup \{(p, y) : y \in W\}$
end
return D_{rsft}

Algorithm 2: Fine-tuning

Input: Base model f_θ with LoRA adapters; dataset D_{rsft}; optimizer $\mathcal{O}$; batch size B; epochs E
Output: Adapted parameters $\theta^\star$

for $e \leftarrow 1$ **to** E **do**
  for *mini-batch $\mathcal{B} \subset D_{rsft}$ of size B* **do**
    compute $\mathcal{L}_{\text{MLE}}(\theta; \mathcal{B}) = -\sum_{(p,y)\in\mathcal{B}} \log p_\theta(y \mid p)$
    backpropagate $\nabla_\theta \mathcal{L}_{\text{MLE}}$; update LoRA parameters using $\mathcal{O}$
  end
end
return $\theta^\star$

12 Datasets

We follow the BLP-2025 Task 2 split (Raihan et al., 2025c) with an organizer-provided *trial* set for format checks, *mHumanEval-Bangla* for development (Raihan et al., 2025a), and *MBPP-Bangla* for held-out evaluation (Raihan et al., 2025b). All prompts are Bangla (`instruction`); unit tests are Python snippets stored in `test_list` and executed in a sandbox. The test split contains hidden tests available only at scoring time.

Each row contains an `id`, a Bangla prompt `instruction`, optional `response` (trial), and Python tests in `test_list`.

Datasets	Row Count	Columns
Trial	74	id, instruction, response, testlist
Dev	400	id, instruction, testlist
Test	500	id, instruction, testlist

Table 1: Dataset splits and schema.

13 Experimental Findings

We varied the inference sampling budget K and evaluated PASS@1 locally:

Sample Budget K	Passes	Total	PASS@1 (%)
10	176	500	35.20
20	192	500	38.40
50	227	500	45.40
100	245	500	49.00

Table 2: Ablation on inference sampling budget K (higher is better). Results reported as number of tasks passed out of 500 and PASS@1 (%).

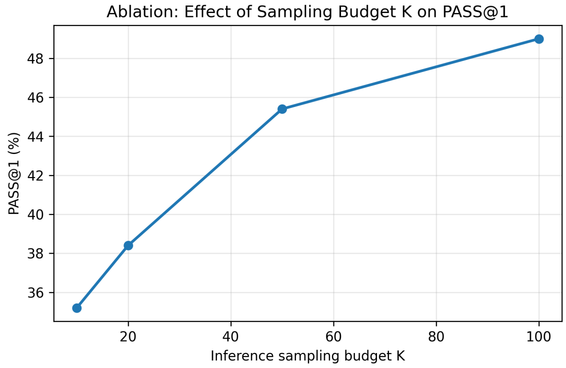

Figure 2: Ablation: accuracy vs. inference sampling budget K.

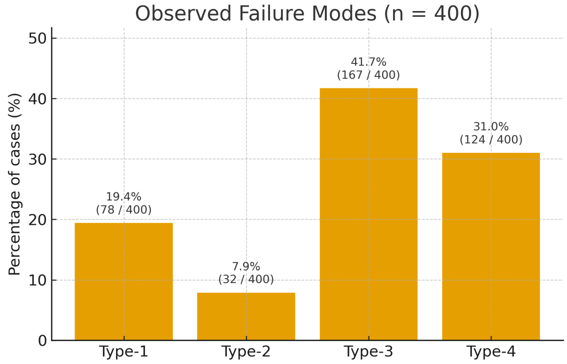

Figure 3: Observed failure modes (Types 1–4) across 400 samples. Labels show percentage and counts.

However, we could achieve a maximum PASS @ 1 score of 31. 6 % (with K = 100) in the online hidden test environment, suggesting that our generated codes were vulnerable to many other edge cases to execute properly.

All experiments ran on an **NVIDIA GeForce RTX 3050** GPU with limited capacity. Unless otherwise stated, we kept mining temperature/top-p, number of mining samples per prompt, LoRA rank and target modules, learning rate, batch size, and epochs constant across the ablation.

Parameter	**Value**
Max Sequence Length	1024
Batch Size (Train/Eval)	16
Gradient Accumulation Steps	4
Max Steps	60
Learning Rate	5×10^{-5}
Weight Decay	0.04
Warmup Steps	10%
Optimizer	AdamW (8-bit)
LR Scheduler	Cosine
Precision	BF16
Seed	3407

Table 3: Hyperparameters used for training, RSFT, and inference.

We observe four recurring classes of failures during development inference for several runs locally:

- **Type-1. Specification misinterpretation:** partial or incorrect adherence to the Bangla prompt (e.g., missing edge conditions, misread constraints).
- **Type-2. I/O contract violations:** mismatch between required and produced interfaces (return vs. print, extra prompts, stray debug output).
- **Type-3. Numerical/algorithmic edge cases:** brittle handling of boundary values (integer vs. float semantics, off-by-one loops, corner-case arithmetic).
- **Type-4. Resource/pathological behavior:** non-terminating or superlinear routines that exceed time/memory limits under hidden tests.

14 Conclusion

We introduced a compact RSFT pipeline for Bangla-to-Python code generation that integrates sampling, sandboxed execution, winner-only supervision, and LoRA-based adaptation. On this task, we observed diminishing returns with larger sampling budgets and identified recurrent failure modes that motivate tighter verification. The approach is simple, reproducible, and safety-conscious via unit-test gating and fuzzed asserts. Future work includes stronger test generation, adaptive mining of K by prompt difficulty, and richer program-analysis signals to improve generalization.

15 Limitations and Ethics

Model quality is bounded by the coverage and rigor of unit tests hence behaviors outside the test distribution may persist, and mining can overfit to artifacts of a particular decoding configuration (e.g., temperature/top-p), favoring shorter or more verbose programs. The generate–execute–select loop also induces selection bias toward easily verifiable solutions and may under-represent semantically correct but slow or non-deterministic code. Although execution occurs in a hardened sandbox, residual risks remain (e.g., resource exhaustion). We therefore adopt defense-in-depth and strict time/memory limits. Due to our limited compute resources, results are further constrained, which restricts hyperparameter sweeps and ablation breadth and may increase variance.

References

Abhik Bhattacharjee, Tahmid Hasan, Wasi Uddin Ahmad, Kazi Samin, Md Saiful Islam, Anindya Iqbal, M. Sohel Rahman, and Rifat Shahriyar. 2022. Banglabert: Language model pretraining and benchmarks for low-resource language understanding evaluation in bangla. *Preprint*, arXiv:2101.00204.

Mark Chen and 1 others. 2021. Evaluating large language models trained on code. *arXiv:2107.03374*.

Charles J. Clopper and Egon S. Pearson. 1934. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413.

Yao Dong and 1 others. 2023. RAFT: Reward ranked finetuning for generative foundation models. *arXiv:2304.06767*.

Edward J Hu and 1 others. 2021. LoRA: Low-rank adaptation of large language models. *arXiv:2106.09685*.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of EMNLP: System Demonstrations*, pages 66–71.

Hung Le and 1 others. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. In *NeurIPS*.

Yujia Li and 1 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Long Ouyang and 1 others. 2022. Training language models to follow instructions with human feedback. *arXiv:2203.02155*.

Romain Paulus, Caiming Xiong, and Richard Socher. 2018. A deep reinforced model for abstractive summarization. In *ICLR*.

Rafael Rafailov and 1 others. 2023. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*.

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).

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence level training with recurrent neural networks. In *ICLR*.

Baptiste Rozière and 1 others. 2023. Code llama: Open foundation models for code. *arXiv:2308.12950*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of ACL*.

Nisan Stiennon and 1 others. 2020. Learning to summarize with human feedback. In *NeurIPS*.

The Unicode Consortium. 2024. Unicode standard annex #15: Unicode normalization forms. Version current at time of writing.

Edwin B. Wilson. 1927. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212.

Eric Zelikman, Yuhuai Wu, Noah D Goodman, and Tomer Holzman. 2022. Star: Bootstrapping reasoning with reasoning. In *NeurIPS*.