DAFFODIL INTERNATIONAL UNIVERSITY

FYDP (Phase-I) Progress Report

REPORTING PERIOD- SPRING 2025

Project Identification:

I. Project Title	**AlgoBugs: Benchmarking LLMs for Corner Case Test Generation in Competitive Programming**	
II. Group Members	Name: MD. MOHIMENUL ISLAM Student ID: 221-15-5725	
III. Supervisor	Name: Sharmin Akter Designation: Assistant Professor	
IV. Co-Supervisor	Name: Saiful Islam Designation: Assistant Professor	
V. Submission Date:	**August 10, 2025**	
VI. Certificate:	"This is to certify that the final year design project work until Phase-I evaluation is held on 10th August 2025, titled 'AlgoBugs: Benchmarking LLMs for Corner Case Test Generation in Competitive Programming', executed by the student mentioned in Sec. II, has been found satisfactory and every section of this report is reflecting the same."	*(Signature of Supervisor & date)*

Project Insights

Thematic Area(s): ***[Just click***	Artificial Intelligence and Machine Learning	✔

the check box]	Data Science and Analytics	☐
	Cybersecurity	☐
	Software Engineering and Development	✔
	Blockchain Technology	☐
	Internet of Things (IoT)	☐
	Computer Networks	☐
	Human-Computer Interaction (HCI)	☐
	Big Data Technologies	☐
	Computer Vision	☐
	Natural Language Processing (NLP)	☐
	Robotics	☐
	Game Development	☐
	Cloud Computing	☐
	Biomedical Computing	☐

	Others *(please specify)*:
Software packages, tools, and programming languages	**Programming Language:** Python, C++ **Environment:** Google Colab, Codeforces Platform **Package List:** 1. NumPy 2. Pandas

	3. Matplotlib 4. Seaborn 5. APIs for LLMs (e.g., GPT-4, Gemini, Claude) 6. Beautifulsoap

CO Description for FYDP-Phase-I

CO	CO Descriptions	PO
CO4	Perform economic evaluation, cost estimation, and apply suitable project management procedures throughout the FYDP lifecycle in the context of developing the "AlgoBugs: Benchmarking LLMs for Corner Case Test Generation in Competitive Programming"	**PO11**
CO6	Select and apply appropriate methodologies, resources, and contemporary engineering/IT tools for prediction, modeling, and solving complex engineering processes for the "AlgoBugs: Benchmarking LLMs for Corner Case Test Generation in Competitive Programming"	**PO5**
CO7	Assess societal, health, safety, legal, and cultural issues and responsibilities in professional engineering practice related to the FYDP problem.	**PO6**
CO10	Operate effectively as an individual and as a member/leader in multidisciplinary teams during FYDP.	**PO9**

1. Project Overview:

1.1 Introduction

The rapid maturation of Large Language Models (LLMs) has transformed their role in software engineering, evolving from simple code completion tools to sophisticated agents capable of complex program synthesis [1], [2]. To rigorously evaluate their advancing capabilities, the research community has increasingly turned to the domain of competitive programming (CP) as a challenging testbed for AI reasoning capabilities [3], [4]. However, the validity of any evaluation in this domain is fundamentally limited by the quality of the test cases used to judge correctness [3], [5]. This critical dependency has catalyzed a new research direction focused on

automated, high-quality test case generation, with LLMs themselves being employed to create more challenging and comprehensive test suites [6], [7]. The frontier of this research has become increasingly adversarial, with recent benchmarks like TestCase-Eval [8] and TCGBench [9] operationalizing the "hacking" phase of programming contests into a formal task. This paper introduces AlgoBugs, a novel benchmark designed to fill this precise diagnostic gap, arguing that to truly understand and improve LLM reasoning, we must move from a single success score to a fine-grained diagnostic profile.

1.2 Background

Early benchmarks for code generation, such as HumanEval [1] and MBPP, focused on function-level synthesis from docstrings. To address their limitations, the research community shifted towards the more demanding domain of competitive programming with benchmarks like APPS [4] and CodeContests [3]. A persistent challenge in this area is data contamination, where models may have been trained on problems present in the evaluation set [11]. This led to a focus on using recent problems to ensure a true test of reasoning. More recently, research has become adversarial, with benchmarks like TestCase-Eval [8] and TCGBench [9] operationalizing the "hacking" phase of contests into a formal task: targeted fault exposure. In this task, an LLM is given a problem description and a specific buggy solution, and it must generate a single test case input that causes the code to fail. While these benchmarks have established that this is an exceptionally difficult task for modern LLMs [8], their evaluation remains monolithic; they can determine *if* a model can find a bug, but not what *kind* of bug it is adept at finding. This is the critical gap AlgoBugs aims to address by providing a systematic, diagnostic evaluation framework.

Table 1.1: Comparison of Relevant Test Generation Benchmarks

Benchmark	**Primary Goal**	**Key Evaluation Task**	**Limitation Addressed by AlgoBugs**
CodeContests+ [5]	Improve RL training dataset quality.	Generate a high-quality, comprehensive test suite.	Lacks a bug taxonomy; evaluation is monolithic and not diagnostic
TestCase-Eval [8]	Evaluate an LLM's test generation ability.	**Fault Exposure:** Generate a single test to break a specific buggy solution.	Also lacks a fine-grained bug classification for diagnostic insights.
TCGBench [9]	Evaluate an LLM's ability to generate a *test case generator*.	Generate a ***generator program*** that produces tests to expose a known bug.	**Also** lacks a bug taxonomy; evaluation is monolithic and not

			diagnostic.

2. Objectives:

i. To create a comprehensive diagnostic benchmark, AlgoBugs, specifically designed for evaluating LLMs on targeted fault exposure tasks in competitive programming contexts.

ii. To develop a sophisticated taxonomy of real-world algorithmic bugs, moving beyond naive classifications to capture subtle errors in areas like dynamic programming and greedy algorithms.

iii. To establish and implement a "gold-standard" data curation workflow for mining recent {Wrong Answer, Accepted} submission pairs from the Codeforces platform to ensure bug authenticity and mitigate data contamination.

3. Methodology/ Requirement Specification:

3.1 Research Design/ Prototype Design

The research is designed to create AlgoBugs, a novel diagnostic benchmark to address a critical gap in LLM evaluation. Unlike existing benchmarks that provide a single success score, our design enables a fine-grained analysis of an LLM's reasoning capabilities against a structured classification of algorithmic errors.

The methodology is divided into two primary stages: (1) The design and curation of the benchmark dataset, and (2) The experimental protocol for evaluating LLMs using this dataset. The design philosophy is inspired by the rigorous standards of established bug-centric datasets like Defects4J [18] and Codeflaws [7]. The project is developed using Python and C++, leveraging the Codeforces platform for data sourcing and various LLM APIs (e.g., GPT-4, Claude 3) for the evaluation stage.

3.2 Data Collection/ Need Assessment

To ensure the highest authenticity and scientific validity, AlgoBugs is populated with real-world bugs mined from the Codeforces platform following a "gold-standard" workflow. This approach mitigates data contamination and ensures the bugs are representative of genuine human errors [11], [12]. The curation protocol for each data point is as follows:

1. **Problem Selection:** Problems are selected from recent Codeforces contests held after the knowledge cutoff dates of the LLMs being evaluated to minimize data contamination [11].

2. **Pair Mining:** Submission pairs are identified from the same user consisting of a "Wrong Answer" (WA) or "Time Limit Exceeded" (TLE) submission followed by an "Accepted" (AC) submission.

3. **Bug Isolation:** The two submissions are compared using a diff utility, and only pairs with

small, localized changes are kept to ensure the bug is clearly isolated.

4. **Subtlety Verification:** The buggy solution is run against all public sample test cases. The data point is only accepted if the buggy solution passes all sample tests, ensuring the bug is a true "corner case" and not a trivial error.

3.3 Analysis Techniques

The primary analysis technique in the curation phase is the manual classification of each isolated bug according to the formal AlgoBugs taxonomy. This taxonomy moves beyond naive classifications to capture the subtle errors that differentiate expert human programmers from automated systems. This classification is what enables the final diagnostic evaluation of the LLMs. The main categories of the taxonomy are:

1. **Algorithmic Design and Logic Errors:** This includes bugs where the overall algorithmic paradigm is correct, but the implementation is flawed, such as an incorrect state representation in dynamic programming (`Incorrect DP State/Transition`) or a heuristic that fails on specific inputs (`Flawed Greedy Choice`). It also covers failures to handle subtle problem requirements, such as a graph being disconnected [16].

2. **Resource and Performance Failures:** This category includes bugs that are not logical errors but performance issues. This includes algorithms vulnerable to worst-case inputs that cause a `Time Limit Exceeded` (TLE) verdict, a known weakness for LLMs [16], [17], as well as bugs that are exposed by inputs forcing the creation of unexpectedly large data structures, leading to a `Memory Limit Exceeded` (MLE) verdict.

3. **Boundary, Edge Case, and Data Type Errors:** This covers classic programming errors, such as those related to minimum/maximum constraint values or integer overflow (`Numeric Boundary/Overflow`), and errors exposed by empty inputs or single-element collections (`Structural Edge Cases`) [16].

3.4 Dataset Structure

Each curated data point in the AlgoBugs benchmark is stored in a structured, machine-readable format to ensure reproducibility and ease of use. The dataset is organized with a master metadata file and individual directories for each problem. This structure ensures clarity and facilitates easy integration with automated evaluation frameworks.

Table 3.1: Structure of a Curated Data Point

File / Directory	Purpose
metadata.json	An index file containing the problem ID, bug category from the taxonomy, and a natural language description of the specific bug.

problem_statement.md	The complete problem description, including constraints and sample input/output, as sourced from Codeforces.
solution.correct.cpp	The "Accepted" C++ source code from the submission pair, which serves as the ground truth solution.
solution.buggy.cpp	The "Wrong Answer" or "Time Limit Exceeded" C++ source code that contains the subtle, isolated bug.

4. Progress Achieved:

4.1 Completed Tasks

Comprehensive Literature Review: Systematically surveyed the state-of-the-art in LLM-driven test case generation and successfully identified a significant research gap for a diagnostic benchmark.

Benchmark Design and Taxonomy Development: Designed the end-to-end methodology for the AlgoBugs benchmark and developed a sophisticated, formal taxonomy of algorithmic bug types.

Data Curation Protocol: Established a "gold-standard" workflow for mining, isolating, and verifying authentic bug-fix pairs from the Codeforces platform.

Initial Data Curation & Proof of Concept: Began mining recent submission pairs and manually classified a preliminary set of bugs as a proof of concept.

Report Documentation: Documented the project's background, methodology, and Phase-I progress in this formal report.

4.2 Results Obtained

The primary results of Phase-I are the successful design of the benchmark and the validation of its novelty, rather than quantitative performance metrics. A key outcome is the established novelty of the project, as the comprehensive literature review successfully confirmed that no existing benchmark systematically evaluates an LLM's ability to generate test cases targeting a pre-defined, formal taxonomy of algorithmic corner cases. Furthermore, a formal bug taxonomy has been created, which serves as the foundational framework for the benchmark's diagnostic capabilities. This taxonomy includes detailed categories for Algorithmic Logic, Resource Failures, and Boundary Cases. Finally, a rigorous, multi-step curation workflow has been finalized, featuring a critical "Subtlety Verification" step to ensure that all curated bugs are true corner cases, thereby guaranteeing a high-quality and challenging benchmark.

5. Challenges Faced:

S.No.	Issues and Challenges	Strategies or Plans
1	**Data Contamination:** The integrity of the evaluation could be compromised as popular LLMs may have been pre-trained on the problems sourced from platforms like Codeforces.	To mitigate this, the curation protocol will exclusively use problems from recent contests held *after* the knowledge cutoff dates of the LLMs being evaluated [11].
2	**Bug Subtlety and Quality:** It is challenging to ensure that collected bugs are genuine "corner cases" and not trivial errors that would be caught by public sample tests.	The "Subtlety Verification" step in the curation protocol is designed to address this. We will only include bug-fix pairs where the buggy code successfully passes all public sample tests provided in the problem description.
3	**Time and Expertise for Classification:** Manually analyzing and classifying each bug according to the detailed taxonomy is time-consuming and requires significant expertise in competitive programming to ensure accuracy.	The classification will be performed by multiple experienced programmers to ensure consistency. **Inter-annotator agreement will be** measured to validate the reliability of the assigned bug categories.
4	**Inherent Task Difficulty for LLMs:** The "targeted fault exposure" task is known to be exceptionally difficult for current LLMs, which could lead to very low success rates across the board.	**Advanced prompting techniques will be** employed such as Chain-of-Thought (CoT) prompting, which has been shown to elicit better reasoning in large language models for complex tasks [16], [17].

6. Next Steps:

This table outlines the immediate, critical path to submitting your paper to the IDAA 2025 conference, along with post-submission plans for the final thesis.

S.No.	Next Task	Estimated completion time
1	**Conduct Baseline Experiments:** Run the core evaluation on 3-4 key LLMs (e.g., GPT-4, Gemini) to generate preliminary results that validate the benchmark's diagnostic utility.	August 22, 2025
2	**Draft and Finalize Conference Paper:** Write the full paper, framing the novel benchmark, taxonomy, and curation protocol as the primary contributions, supported by the baseline experimental results.	August 27, 2025
3	**Internal Review and Submission:** Submit the finalized draft to the supervisor for review and submit the paper to the IDAA 2025 conference portal by the deadline.	August 31, 2025
4	**Refine and Expand Experiments:** Post-submission, continue to refine the dataset and explore more advanced prompting techniques to generate enhanced results for the final thesis defense.	October 15, 2025

7. Updated Timeline:

Tasks	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10
Conduct Baseline Experiments										
Draft and Finalize IDAA Conference Paper										
Internal Review and Submission										
Refine and Expand Experiments										

Estimated Work Period	
Actual Work Period	

8. Resources Utilized:

The development and evaluation of the AlgoBugs benchmark rely on a combination of data, computational, and software resources.

- **Data Sources:** The primary data source is the Codeforces platform, which is used for mining real-world, authentic bug-fix submission pairs that form the core of the dataset.
- **Computational Resources:** The experimental phase requires access to the official APIs for state-of-the-art proprietary language models such as GPT-4, Claude 3, and Gemini Pro.
- **Software and Libraries:**
    - The project's evaluation pipeline and analysis scripts are developed primarily in Python.
    - Essential Python libraries include `NumPy` and `Pandas` for data manipulation, `Matplotlib` and `Seaborn` for results visualization, and `Scikit-learn` for calculating evaluation metrics..pdf].
    - Version Control is managed using Git, with repositories hosted on GitHub to ensure code integrity and track development progress.

9. Project Management and Financial Analysis:

Table 9.1: Estimated Cost for the Project

SN	Components	Estimated Cost (BDT)
01.	LLM API Usage Costs (for experiments)	5,000 - 8,000
02.	Cloud Data Storage	1,000 - 1,500
03.	Conference Registration Fee (IDAA 2025, if accepted)	6,000 - 10,000
04.	Documentation and Thesis Printing/Binding	1,500 - 2,000
05.	Contingency (10% of total)	1,350 - 2,150
Total Estimated Cost		14,850 - 23,650

10.Future Considerations:

The immediate focus is the successful submission of a research paper to the IDAA 2025 conference by the August 31st deadline. Post-submission, the next phase of the project will involve scaling up the experiments for the final thesis. A key consideration is a deeper comparative analysis between the performance of proprietary models and open-source models like StarCoder and CodeT5+. A primary future research direction will be to explore the fine-tuning of these open-source models on specific bug categories from the AlgoBugs dataset. Should the paper be accepted, a critical task will be preparing the final camera-ready version, ensuring it meets all scope and quality requirements for inclusion in the IEEE Xplore Digital Library.

11.Conclusion:

This report has outlined the progress of Phase-I, which has successfully established a clear research gap and resulted in the complete design of a novel diagnostic benchmark, "AlgoBugs". The project's core contributions—a sophisticated bug taxonomy and a "gold-standard" data curation protocol—are fully defined. The immediate objective is to finalize a research paper based on these contributions, supported by a preliminary experimental evaluation, for submission to the IDAA 2025 conference. This work provides a strong foundation for the expanded analysis required for the final thesis and contributes a valuable new tool to the research community.

References

[1] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al., "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[2] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, et al., "Deepseek-coder: When the large language model meets programming-the rise of code intelligence," *arXiv preprint arXiv:2401.14813*, 2024.

[3] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, et al., "Competition-level code generation with alphacode," *Science*, vol. 378, no. 6624, pp. 1092-1097, 2022.

[4] D. Hendrycks, S. Basart, S. Kadavath, M. Mazeika, A. Arora, E. Guo, C. Burns, S. Puranik, H. He, D. Song, et al., "Measuring coding challenge competence with apps," *arXiv preprint arXiv:2105.09938*, 2021.

[5] Z. Wang, J. Zhang, Y. Wang, and G. Li, "Codecontests+: High-quality test case generation for competitive programming," *arXiv preprint arXiv: 2506.05817*, 2025.

[6] Z. Wang, J. Zhang, Y. Wang, and G. Li, "Testeval: Benchmarking large language models for test case generation," *Findings of the Association for Computational Linguistics: NAACL 2025*, 2025.

[7] S. H. Tan, A. Roychoudhury, X. Shi, and Z. Cai, "Codeflaws: a programming competition benchmark for evaluating automated program repair tools," in *2017 IEEE/ACM 39th International Conference on Software*

Engineering Companion (ICSE-C), pp. 89-91, IEEE, 2017.

[8] Z. Yang, Y. Cao, J. Zhang, Y. Wang, and G. Li, "Can llms generate high-quality test cases for algorithm problems? testcase-eval: A systematic evaluation of fault coverage and exposure," *arXiv preprint arXiv: 2506.12278*, 2025.

[9] Y. Cao, Z. Yang, J. Zhang, Y. Wang, and G. Li, "Can llms generate reliable test case generators? a study on competition-level programming problems," *arXiv preprint arXiv:2506.06821*, 2025.

[10] Z. Wang, J. Zhang, Y. Wang, and G. Li, "An empirical study on the robustness of llm-generated code," *arXiv preprint arXiv:2503.20197*, 2025.

[11] J. Huang, X. Li, T. Shi, Q. Zhou, D. Chen, and Z. Wu, "Competition-level problems are effective llm evaluators," *arXiv preprint arXiv: 2402.15886*, 2024.

[12] M. M. A. Haque, W. U. Ahmad, I. Lourentzou, and C. Brown, "Fixeval: Execution-based evaluation of program fixes for competitive programming problems," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1-14, 2023.

[13] N. Dong, "A survey of prompt engineering for large language models," *Medium*, September 2024.

[14] PromptPanda, "Few shot prompting explained: A guide," *promptpanda.io*, 2025.

[15] L. Prompting, "Shot-based prompting: Zero-shot, one-shot, and few-shot prompting," *learnprompting.org*, 2025.

[16] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, F. Xia, Q. Adams, and P. Vincent, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, 2022.

[17] F. Li et al., "Uncertainty-guided chain-of-thought for code generation with llms," *arXiv preprint arXiv:2503.15341*, 2025.

[18] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A database of existing faults to enable controlled testing studies for Java programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, 2014, pp. 437–440.