Semester Long Internship Report

Autumn, 2025

On

Benchmarking Large Language Models for Logical Error Detection

Submitted by

Tanishi Rai

VIT Bhopal University, Bhopal

Under the guidance of

Dr. Kushal Shah

and

Prof. Prabhu Ramachandran

FOSSEE, Indian Institute of Technology Bombay

March 2026

Declaration

I hereby declare that this internship report, entitled *"Benchmarking Large Language Models for Logical Error Detection in Python Programming Education"*, is an original and authentic work submitted in partial fulfilment of the requirements for the FOSSEE Semester-Long Internship (Part-Time), conducted by the Free/Libre and Open Source Software for Education (FOSSEE) team at the Indian Institute of Technology Bombay.

All content herein is expressed in my own words, except where explicit references have been made to the ideas or works of others, which have been duly acknowledged and cited. No part of this submission has been plagiarised, misrepresented, fabricated, or falsified, nor has it been submitted to any other institution for the award of any degree or academic credit.

Name: Tanishi Rai

Internship: FOSSEE Semester-Long Internship, Autumn 2025

Guide: Dr. Kushal Shah

Date: April 2026

Abstract

This report presents the work carried out during a part-time semester internship with the FOSSEE team at IIT Bombay, focusing on a rigorous comparative benchmark of six Large Language Models on their ability to classify logical errors in student-written Python programs drawn from the Yaksh learning platform.

The study employs an eight-category Structural-Local (SL) error taxonomy covering: Loop Condition (A), Condition Branch (B), Statement Integrity (C), Output/Input Format (D), Variable Initialization (E), Data Type (F), Computation (G), and a NONE category. One hundred Python code snippets from the Yaksh platform serve as the evaluation dataset. Models are evaluated under two paradigms: a **Single-Label** mode (identify the single most dominant error) and a **Multi-Label** mode (identify all applicable error types). Both modes are run twice to measure run-to-run stability. Performance is assessed using four complementary metrics — Precision (Case A), Recall (Case B), Any-Overlap (Case C), and Jaccard Index (Case D).

Key findings show significant variation in model accuracy (15–40%) and substantial performance differences between single-label (average 53.3%) and multi-label (average 8.7% exact match) modes. GPT-5.2 emerges as the top overall performer, while GPT-OSS-120B is the leading open-weight model. The results demonstrate that fully automated logical error detection at production-quality accuracy remains an open problem, and that human oversight is essential for complex reasoning tasks in educational settings.

Acknowledgements

I would like to express my sincere gratitude to the **FOSSEE team at IIT Bombay** for providing me with the opportunity to participate in the Semester-Long Internship Programme. The collaborative environment throughout the programme made this a truly enriching experience.

I am deeply thankful to my mentors, **Dr. Kushal Shah** and **Prof. Prabhu Ramachandran**, for their guidance, thoughtful feedback, and continuous encouragement throughout the internship. Their insights were instrumental in framing the research questions and shaping the direction of this project.

I would also like to acknowledge **Ms. Usha Viswanathan**, **Ms. Vineeta Ghavri**, and **Mr. Prathamesh Salunke**, and other team members at FOSSEE, for their support during the programme. They were always approachable and helpful whenever questions or matters arose, and their assistance is sincerely appreciated

I am grateful for access to the **Yaksh online programming platform** dataset maintained by the FOSSEE team, which provided an authentic and large-scale foundation for this study.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Background and Motivation

Programming education at scale presents a fundamental challenge: providing timely, precise feedback on logical errors to a large number of learners. Unlike syntax errors, which compilers and interpreters flag immediately, logical errors produce code that runs without crashing yet yields incorrect results. They are far harder to detect automatically because their presence depends on the *intent* of the program rather than its surface-level structure.

The advent of Large Language Models (LLMs) with strong code-understanding capabilities has opened a new avenue for automated pedagogical feedback. Models such as GPT, Claude, Gemini, and their open-weight counterparts have demonstrated impressive performance on a variety of code-related tasks. However, their behaviour on the nuanced task of *classifying the type* of logical error present in student code — as opposed to merely detecting that an error exists — is not yet well-characterised.

Prior work on feedback types in Automated Programming Assessment Systems (APAS) showed that while students perceive unit-test feedback as most useful, AI-generated feedback leads to significantly better learning outcomes [1]. This performance–perception gap directly motivates improving the reliability of LLM-generated explanations before wider deployment in APAS.

The FOSSEE project at IIT Bombay is engaged in developing open-source tools to support technical education in India. In this context, the present internship project was motivated by the question: *can state-of-the-art LLMs reliably serve as automated logical error classifiers in a programming education pipeline?*

1.2 Problem Statement

Existing benchmarks for LLMs on code primarily focus on code generation, bug fixing, or binary correctness prediction. The specific task of multi-class logical error classification using a structured pedagogical taxonomy is less explored. Further, most evaluations test a single model in isolation; comparative studies across proprietary and open-weight models under identical conditions, with reproducible prompts and a standardised dataset, are rare.

This project addresses these gaps by:

1. Designing and running a reproducible inference pipeline across six leading LLMs under identical conditions.
2. Evaluating both constrained (single-label) and unconstrained (multi-label) classification modes, each run twice for stability analysis.
3. Analysing inter-model agreement, behavioural profiles, and systematic biases.
4. Measuring performance against human-annotated ground truth using four complementary evaluation metrics.

1.3 Objectives

The specific objectives of this internship were:

1. To conduct a focused literature review of existing benchmarks and identify gaps motivating this research.
2. To design an eight-category Structural-Local error taxonomy suited to logical errors in introductory Python programs.
3. To curate a dataset of 100 manually annotated Python programs with ground truth logical error labels from the Yaksh platform.
4. To build a robust, reproducible benchmarking pipeline that calls multiple LLMs and saves results incrementally.
5. To design effective single-label and multi-label prompt templates grounded in the taxonomy.
6. To evaluate model outputs using four complementary metrics: Precision, Recall, Any-Overlap, and Jaccard.
7. To analyse consensus, stability, behavioural profiles, and systematic biases across models.
8. To document findings in a reusable analysis notebook and formal report.

1.4 Repository

The primary team repository integrating results across all three taxonomies is publicly available at:

https://github.com/tanishirai/Yaksh-Benchmarking

1.5 Technologies and Tools Used

- **Language:** Python 3
- **LLM APIs:**
  - `openai` — GPT-5.2 and GPT-OSS-120B
  - `anthropic` — Claude Sonnet 4.5
  - Gemini 2.5 Flash, DeepSeek-V3.2, Qwen3-Coder via their respective APIs
- **Data Formats:** Structured JSON; Excel (.xlsx) and CSV for analysis
- **Tools:** VS Code, Overleaf, Git, GitHub
- **Dataset:** Yaksh Online Programming Portal, IIT Bombay

2 Literature Review

2.1 Overview

Prior to the implementation work, a literature review was conducted to understand existing benchmarks for evaluating LLM performance on code-related tasks, with a specific focus on feedback quality in educational settings. Five benchmarks were reviewed. The central finding was that **no existing benchmark specifically evaluates LLM logical error type classification on authentic student code** — a gap that directly motivated the research design of this project.

Table 2.1: Comparative Overview of Reviewed Benchmarks

Benchmark	**Primary Focus**	**Evaluation**	**Key Limitation**
TutorBench (2025)	STEM tutoring	LLM-judge rubric	Not code-debugging specific
MathTutorBench (2025)	Math tutoring	Reward model	Math only; no code
DebugBench (2024)	Bug repair	Pass/fail tests	No feedback quality; synthetic bugs
Dean of LLM Tutors (2024)	CS feedback quality	Accuracy/F1 vs. human	Small, synthetic, CS-only
Survey on Feedback in APAS (2024)	Feedback type comparison	Student surveys	Not an LLM quality benchmark

2.2 Key Benchmarks and Findings

TutorBench evaluates LLMs as tutors across STEM domains using a 16-dimension rubric scored by an LLM-judge. **MathTutorBench** tests mathematics tutoring quality using a pedagogy-aware reward model. **DebugBench** measures code repair on 4,253 LeetCode-style programs via execution-based pass rates — but only checks whether code is fixed, not whether any explanation is correct, and uses synthetically injected bugs. **Dean of LLM Tutors** is closest in spirit to this project, benchmarking

AI tutor feedback across 16 quality dimensions against human labels, though its dataset is small and entirely synthetic.

The **APAS feedback survey** [1] is the most practically relevant benchmark reviewed. It compared compiler, unit-test, and AI-generated feedback in a controlled 90-minute study with 212 students across two universities. Despite unit-test feedback being rated more useful by students, AI feedback (GPT-3.5-Turbo) produced the highest correctness scores (65.44% vs. 57.52% and 41.41%), and led to more submission attempts and longer reflection time between attempts. This performance–perception gap directly motivates improving the reliability of LLM-generated explanations before wider deployment.

2.3 Identified Gaps

Four critical gaps motivated the research direction of this internship:

1. **No benchmark for logical error type classification.** All reviewed work targets code repair, general tutoring, or feedback style comparison. None evaluates whether LLMs can correctly identify the *type* of logical error present in student code.

2. **No benchmark uses authentic student submissions.** Synthetic bugs and synthetic student work do not capture the ambiguity and diversity of real student code.

3. **No cross-model disagreement analysis.** No existing work studies how and why different LLMs disagree on error classification.

4. **No multi-label evaluation framework** with partial overlap metrics has been applied to logical error classification in this domain.

3 Error Taxonomy

3.1 Structural-Local (SL) Taxonomy

This project adopts an eight-category Structural-Local (SL) taxonomy designed to provide fine-grained, instructionally meaningful categories for logical errors in Python programs. Each category corresponds to a distinct aspect of program structure and is intended to capture a different type of student mistake.

Table 3.1: The Eight-Category SL Error Taxonomy Used in This Study

ID	Category	Group	Description
A	Loop Condition	Control Flow	Incorrect loops in `for`/`while` condition statements
B	Condition Branch	Control Flow	Incorrect expression in the `if` condition
C	Statement Integrity	Code Structure	Statement lacks a part of its logical structure
D	Output/Input Format	Data Handling	Incorrect `input()`/`print()` statement
E	Variable Init	Data Handling	Incorrect declaration or initialisation of variables
F	Data Type	Data Handling	Incorrect data type usage
G	Computation	Computation	Incorrect basic math symbols or operators
NONE	No Error	—	No logical error present (code is correct)

3.2 Higher-Level Groupings

The eight categories can be organised into four higher-level conceptual groups for aggregate analysis:

- **Control Flow Errors (A, B):** Errors that disrupt execution flow via incorrect loop conditions or conditional expressions.

- **Code Structure Errors (C):** Incomplete or malformed code statements that break logical structure.
- **Data Handling Errors (D, E, F):** Errors in input/output format, variable initialisation, or data type usage.
- **Computation Errors (G):** Incorrect mathematical operators or expressions.

3.3 Dataset Characteristics

Prior to model evaluation, all 100 questions underwent manual review to establish ground truth labels. Table 3.2 shows the frequency distribution of error categories. The total category assignments (179) exceed 100 because multi-label questions contain multiple error types; the average is 1.79 labels per question.

Table 3.2: Error Category Distribution in Manual Labels (100 Questions)

ID	Category	Count	Percentage
C	Statement Integrity	51	51.0%
D	Output/Input Format	49	49.0%
G	Computation	31	31.0%
NONE	No Error	20	20.0%
F	Data Type	10	10.0%
B	Condition Branch	8	8.0%
A	Loop Condition	5	5.0%
E	Variable Init	5	5.0%
Total assignments		179	—

4 Methodology

4.1 Dataset

The evaluation dataset comprises **100 Python programming questions** from the Yaksh learning platform. Each item contains a natural-language problem description and a student-submitted answer. Human-annotated ground truth (GT) labels were produced for each snippet using the SL taxonomy. GT labels are multi-label: a snippet may be labelled with multiple co-occurring error types (e.g., C and D simultaneously). Where a snippet contains no logical error, it is labelled NONE.

4.2 Models Evaluated

Six leading LLMs were selected, representing a mix of closed-source and open-weight architectures. Table 4.1 summarises them.

Table 4.1: LLMs Evaluated in the Benchmark

Model	**Provider**	**Type**	**API**
Claude Sonnet 4.5	Anthropic	Closed-source	`anthropic`
GPT-5.2	OpenAI	Closed-source	`openai`
Gemini 2.5 Flash	Google	Closed-source	Gemini API
DeepSeek-V3.2	DeepSeek	Open-weight	DeepSeek API
Qwen3-Coder	Alibaba	Open-weight	Qwen API
GPT-OSS-120B	OpenAI	Open-weight	`openai`

4.3 Classification Modes

Two prompt paradigms were evaluated:

- **Single-Label Mode:** The model must identify the single most dominant error category. Output is constrained to exactly one label from {A, B, C, D, E, F, G, NONE}.
- **Multi-Label Mode:** The model must identify all applicable error types. Output is a comma-separated list (e.g., C, D, G). If no error is present, the model outputs NONE alone.

Each mode was run **twice** on the full 100-question dataset (Run 1 and Run 2) to measure run-to-run stability. This yielded 6 models $\times$ 2 modes $\times$ 2 runs $\times$ 100 questions = 2,400 individual API calls.

4.4 Prompt Design

The prompts were structured, minimal, and taxonomy-driven. Both prompts include: a role declaration, a task description, the full taxonomy in XML format, a reasoning instruction block, strict output format rules, and the student code snippet. An abbreviated view of the single-label prompt is shown in Listing 4.1.

```
<role>Expert Python Programming Assistant</role>

<task>
Identify the SINGLE most dominant logical error in the
provided Python code based on the taxonomy below.
</task>

<taxonomy>
  <category label="A">
    <title>Loop Condition</title>
    <description>Incorrect loops in for/while condition
    statements.</description>
  </category>
  <!-- ... all 8 categories ... -->
</taxonomy>

<output_rules>
- Output ONLY the single label (A, B, C, D, E, F, G, or NONE).
- STRICT: Any text other than the label is a failure.
</output_rules>

<code_to_analyze>
{code_input}
</code_to_analyze>
```

Listing 4.1: Single-label prompt template (abbreviated).

The multi-label prompt is identical except the task and output rules instruct the model to output a comma-separated list of all applicable labels.

4.5 Evaluation Metrics

Standard binary accuracy is insufficient for multi-label evaluation. Let GT denote the ground-truth label set and $Pred$ the predicted label set for a given question. Four complementary cases are defined:

Table 4.2: Evaluation Cases Used for Scoring Model Predictions

Case	**Name**	**Definition**
A	Precision (Subset)	$Pred \subseteq GT$. All predicted labels are correct; measures false-positive control.
B	Recall (Superset)	$GT \subseteq Pred$. All ground-truth labels are covered; measures false-negative control.
C	Any Overlap	$Pred \cap GT \neq \emptyset$. At least one correct label is predicted.
D	Jaccard Index	$J = \frac{\lvert Pred \cap GT \rvert}{\lvert Pred \cup GT \rvert}$. Penalises both omissions and hallucinations equally. Primary metric.

The Jaccard Index (Case D) is the primary accuracy metric as it provides the most balanced view. A cleaning pipeline was applied to all raw outputs before evaluation: empty or "None" responses were unified under NONE; multi-label outputs were sorted and deduplicated; a regex step extracted labels from any surrounding text.

5 Results and Discussion

5.1 Overall Model Performance

Table 5.1 presents aggregate accuracy averaged across all runs and prompting strategies for each model.

Table 5.1: Overall Model Performance Rankings

Model	**Accuracy (%)**	**Rank**
GPT-5.2	40.5	1
GPT-OSS-120B	40.0	2
Claude Sonnet 4.5	35.0	3
Gemini 2.5 Flash	33.3	4
Qwen3-Coder	23.5	5
DeepSeek-V3.2	16.8	6
Mean	31.5	—

GPT-5.2 achieves the highest average accuracy (40.5%), correctly classifying approximately 40 out of 100 questions. DeepSeek-V3.2 performs weakest (16.8%). The 23.8 percentage-point spread indicates substantial variation in capabilities across models. All models are best suited to assistive roles with human oversight rather than fully autonomous classification.

5.2 Prompting Strategy Comparison

Single-label and multi-label prompting are compared in Table 5.2.

Table 5.2: Single-Label vs. Multi-Label Accuracy by Model

Model	**Single (%)**	**Multi (%)**	**Difference (pp)**
GPT-5.2	69.5	11.5	−58.0
GPT-OSS-120B	65.5	14.5	−51.0
Claude Sonnet 4.5	57.5	12.5	−45.0
Gemini 2.5 Flash	53.5	13.0	−40.5
DeepSeek-V3.2	29.5	4.0	−25.5
Qwen3-Coder	45.5	1.5	−44.0
Average	53.3	8.7	−44.6

The dramatic single-to-multi performance drop (44.6 pp average) stems from three factors: (1) combinatorial complexity — predicting the correct subset from 8 categories is exponentially harder than selecting one of 8; (2) precision-recall tension — models must balance avoiding false positives with catching false negatives; (3) label co-occurrence uncertainty — models struggle to determine which errors genuinely co-exist. **Single-label prompting is recommended as the default** for deployment (53.3% vs. 8.7% multi-label exact match).

5.3 Consistency and Reliability

5.3.1 Overall Agreement Rates

Table 5.3 shows inter-run agreement rates (how often a model gives the same answer across Run 1 and Run 2).

Table 5.3: Inter-Run Agreement Rates (Average Across Both Modes)

Model	**Agreement Rate (%)**
GPT-5.2	75.5
GPT-OSS-120B	64.5
Qwen3-Coder	63.5
Claude Sonnet 4.5	55.0
Gemini 2.5 Flash	55.0
DeepSeek-V3.2	44.5
Average	59.7

5.3.2 Agreement Quality (Single-Label Mode)

To understand whether consistency indicates reliability, both runs were compared against ground truth labels. Table 5.4 breaks down what happened in single-label mode.

Table 5.4: Agreement Quality Breakdown — Single-Label Mode

Model	Same Answer	Changed	Same & Correct	Same but Wrong
GPT-5.2	85	15	70	15
GPT-OSS-120B	75	25	63	12
Qwen3-Coder	78	22	46	32
Claude Sonnet 4.5	66	34	59	7
Gemini 2.5 Flash	67	33	54	13
DeepSeek-V3.2	56	44	28	28

Claude shows the best agreement quality (89% of consistent answers are correct). **DeepSeek** is worst: when it agrees with itself (56 times), it is only correct half the time. **Qwen** shows a concerning pattern — agreeing 78 times but making consistent errors 32 times (41% of agreements are wrong).

5.4 Pairwise Model Agreement

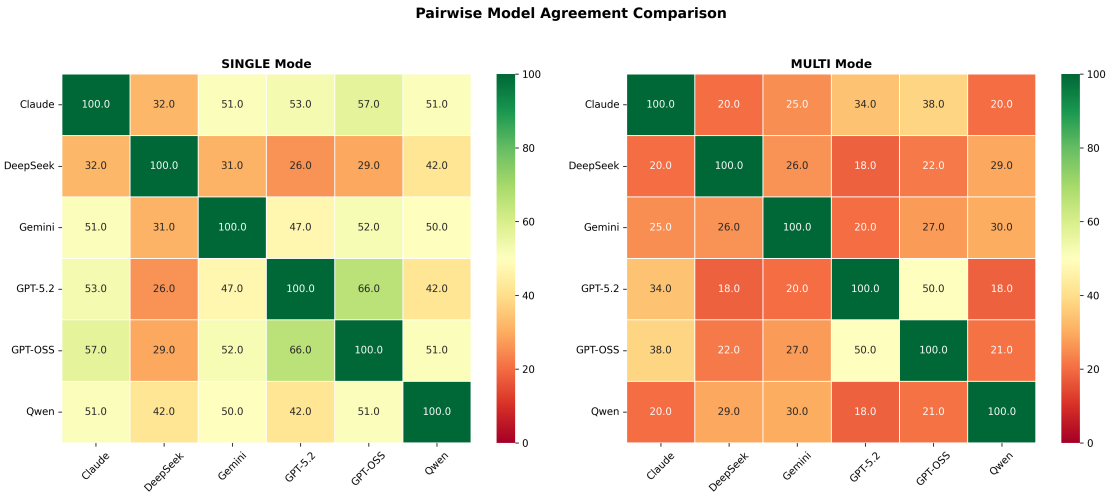

Figure 5.1: Pairwise model agreement matrices for single-label mode (left) and multi-label mode (right). Diagonal values are 100%. Higher off-diagonal values indicate similar prediction patterns.

Tables 5.5 and 5.6 present the pairwise agreement matrices numerically.

Table 5.5: Pairwise Agreement Matrix — Single-Label Mode (%)

	Claude	**DeepSeek**	**Gemini**	**GPT-5.2**	**GPT-OSS**	**Qwen**
Claude	100	32	51	53	57	51
DeepSeek	32	100	31	26	29	42
Gemini	51	31	100	47	52	50
GPT-5.2	53	26	47	100	66	42
GPT-OSS	57	29	52	66	100	51
Qwen	51	42	50	42	51	100

Table 5.6: Pairwise Agreement Matrix — Multi-Label Mode (%)

	Claude	**DeepSeek**	**Gemini**	**GPT-5.2**	**GPT-OSS**	**Qwen**
Claude	100	20	25	34	38	20
DeepSeek	20	100	26	18	22	29
Gemini	25	26	100	20	27	30
GPT-5.2	34	18	20	100	50	18
GPT-OSS	38	22	27	50	100	21
Qwen	20	29	30	18	21	100

GPT-5.2 and GPT-OSS-120B form the highest-agreement pair in both modes (66% single, 50% multi), reflecting shared architectural lineage. DeepSeek shows the lowest agreement with all other models — as low as 26% in single-label mode — indicating fundamentally different error detection behaviour. In multi-label mode, all off-diagonal values collapse dramatically, with most pairs falling below 30%.

5.5 Closed-Source vs. Open-Weight Comparison

Table 5.7: Average Accuracy: Closed-Source vs. Open-Weight Models

Mode	**Closed-Source (%)**	**Open-Weight (%)**	**Gap (pp)**
Single-label	61.0	45.7	+15.3
Multi-label	11.0	6.3	+4.7
Average	36.0	26.0	+10.0

Closed-source models hold a 10 pp average advantage. However, **GPT-OSS-120B consistently leads among open-weight models** and often matches closed-source performance, suggesting that model scale and instruction-tuning quality are stronger performance drivers than weight accessibility.

5.6 Ensemble Methods

An ensemble approach combines predictions from all six models using majority voting. Table 5.8 compares ensemble performance to individual model averages.

Table 5.8: Ensemble Performance vs. Individual Models

Mode	Avg Individual (%)	Ensemble (%)	Improvement (pp)
Single-label	53.3	57.0	+3.7
Multi-label	8.7	11.0	+2.3

In single-label mode, 11 questions achieved unanimous consensus (all correct) and 29 had a clear majority (correct). In multi-label mode, zero questions achieved perfect consensus — models split on 87% of questions, making majority voting nearly ineffective.

5.7 Multi-Label Partial Overlap Analysis

Table 5.9 presents model performance across all four partial overlap metrics, averaged across both runs.

Table 5.9: Multi-Label Partial Overlap Performance (Averaged Across Runs)

Model	Case A (%)	Case B (%)	Case C (%)	Jaccard
GPT-5.2	31.7	45.2	96.2	0.556
Claude Sonnet 4.5	32.7	38.5	93.3	0.511
GPT-OSS-120B	35.6	28.8	90.4	0.475
Gemini 2.5 Flash	28.8	10.6	73.1	0.325
DeepSeek-V3.2	13.5	26.9	70.2	0.304
Qwen3-Coder	7.7	24.0	76.0	0.291
Average	25.0	29.0	83.2	0.410

GPT-5.2 dominates across all four metrics, particularly excelling in any-overlap (Case C = 96.2%) and Jaccard (0.556). The dramatic 76.0 pp gap between exact match (7.2%) and any-overlap (83.2%) reveals that models typically identify at least one correct error even when predictions are incomplete. The Jaccard score is the most informative single metric, as it penalises both omissions and hallucinations equally.

5.8 Category-Specific Performance Analysis

5.8.1 Single-Label Performance

Table 5.10 shows single-label category performance ranked by average F1-score across all six models.

Table 5.10: Single-Label Category Performance (Avg. F1-Score Across All Models)

Rank	Category	F1 (%)	Recall (%)	Precision (%)
1	NONE (No Error)	67.2	69.2	66.0
2	B (Condition Branch)	48.7	64.8	45.5
3	C (Statement Integ.)	48.1	37.3	69.7
4	F (Data Type)	26.7	20.0	40.8
5	D (I/O Format)	22.1	13.9	77.4
6	G (Computation)	21.8	14.5	66.6
7	E (Variable Init)	21.4	23.3	21.4
8	A (Loop Condition)	19.1	30.0	15.1
Avg. (excl. NONE)		29.7	29.1	47.8

5.8.2 Multi-Label Performance

Table 5.11: Multi-Label Category Performance (Avg. F1-Score, NONE Excluded)

Rank	Category	F1 (%)	Recall (%)	Precision (%)
1	C (Statement Integ.)	61.4	56.3	69.5
2	G (Computation)	55.8	60.2	53.0
3	D (I/O Format)	55.6	46.3	81.9
4	F (Data Type)	48.3	61.7	43.8
5	B (Condition Branch)	46.2	90.7	32.0
6	A (Loop Condition)	25.6	76.7	15.6
7	E (Variable Init)	12.9	73.3	7.1
Average		43.7	66.5	43.3

5.8.3 Comprehensive Visual Analysis

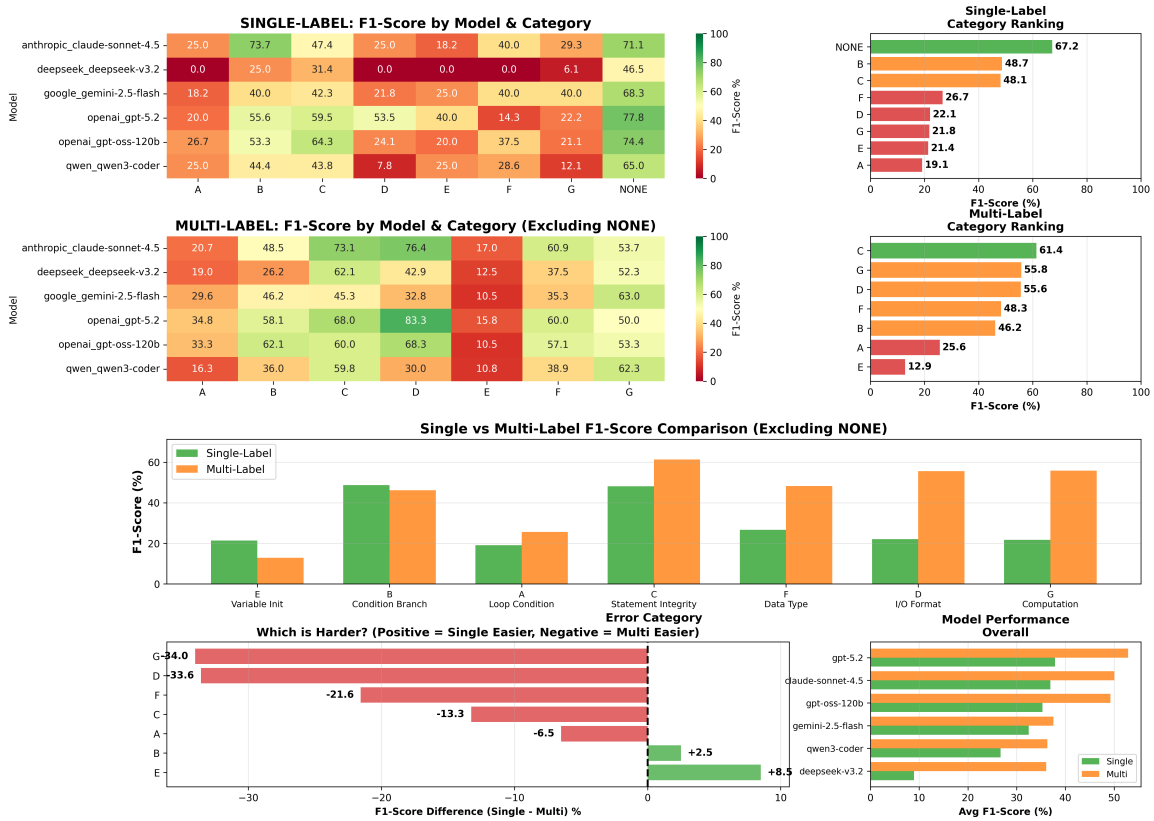

Figure 5.2: Comprehensive category performance analysis — single vs. multi-label. Heatmaps show F1-scores by model and category; bar charts compare single and multi-label performance per category and show which categories benefit from multi-label prompting.

5.8.4 Key Observations

Four of seven categories (C, D, F, G) show *higher* F1-scores in multi-label mode than single-label mode — a surprising reversal. This occurs because these categories co-occur frequently, and the multi-label prompt allows models to assign them together. Categories E and B decline under multi-label, suggesting that fine-grained single-mistake categories are harder to isolate when models are permitted to over-predict.

5.9 Manual Label Verification

During analysis, systematic discrepancies emerged between model consensus and manual labels. A verification process identified **12 samples where all 6 models agreed but disagreed with the human label**. Table 5.12 summarises the findings.

Table 5.12: Manual Label Verification Summary

Category	**Samples Identified**	**Avg. Jaccard**
All models agree (vs. human)	12	0.234
Partial matches (Jaccard < 0.30)	477 instances	0.180
Complete mismatches (Jaccard $= 0$)	14 instances	0.000

In several cases, model consensus appeared justifiable upon review — suggesting potential annotation inconsistencies in the human baseline. These cases warrant re-evaluation before using this dataset as a formal public benchmark.

6 Conclusions and Future Work

6.1 Summary of Findings

This internship project produced a rigorous, reproducible benchmark for evaluating six LLMs on logical error classification in student Python code. The following conclusions are supported by the data:

1. **Single-label prompting substantially outperforms multi-label.** The average accuracy drop of 44.6 pp when switching to multi-label reflects combinatorial complexity and precision-recall tension. Single-label prompting (53.3% average) is recommended for most deployment contexts.
2. **GPT-5.2 is the top performer** with 40.5% overall accuracy, 75.5% inter-run agreement, and the highest Jaccard score (0.556) in multi-label evaluation. GPT-OSS-120B is a strong open-weight alternative.
3. **Category performance varies substantially.** NONE is the easiest category in single-label mode (67.2% F1); Variable Init (E) and Loop Condition (A) are consistently hardest. Four categories (C, D, F, G) improve in multi-label mode.
4. **Closed-source models hold a 10 pp average advantage,** but GPT-OSS-120B matches or exceeds closed-source performance, suggesting model scale matters more than proprietary access.
5. **Ensemble voting provides modest gains** (+3.7 pp single-label) but is nearly ineffective in multi-label mode (87% split decisions).
6. **Human oversight remains essential.** With average accuracy of 31.5% and near-zero multi-label exact match, current models are suitable as assistive tools requiring human verification rather than autonomous classifiers.

6.2 Limitations

- The dataset of 100 questions, while sufficient for initial analysis, is small; larger evaluation would improve statistical power.
- The SL taxonomy may not cover all logical error types encountered in more advanced exercises.

- Human annotations were produced by a single annotator without inter-annotator agreement measurement, introducing potential subjectivity.
- Model outputs were collected at a fixed point in time; version updates may affect reproducibility.

6.3 Future Work

1. **Larger datasets:** Expand to hundreds or thousands of questions across multiple difficulty levels and programming languages.
2. **Few-shot and Chain-of-Thought prompting:** Integrate structured reasoning into prompts to improve fine-grained category discrimination.
3. **Fine-tuning:** Fine-tune open-weight models on annotated logical error data to reduce systematic biases.
4. **Integration into Yaksh:** Embed the best-performing model as an automated feedback module within the Yaksh platform.
5. **Inter-annotator agreement:** Develop a consensus annotation process to produce higher-quality ground truth labels for ambiguous cases.

6.4 Personal Reflection

This internship provided a structured opportunity to combine prompt engineering, API-based software development, statistical analysis, and technical writing into a cohesive research project.

On the **technical side**, I gained hands-on experience designing and running large-scale LLM inference pipelines, handling API rate limits and incremental result saving, and building reproducible analysis notebooks using `pandas`, `json`, `matplotlib/seaborn` and `numpy`.

On the **research side**, working within a team across three complementary taxonomies required developing comfort with ambiguity and subjective annotation. Designing evaluation metrics that meaningfully capture partial correctness (Cases A through D) was a particularly instructive exercise, and collaborating with Harshit and Yuvraj on the shared dataset provided valuable perspective on how taxonomy design choices directly affect measurable outcomes.

References

[1] A Survey on Feedback Types in Automated Programming Assessment Systems, core paper source.

[2] FOSSEE Team, IIT Bombay, "Yaksh: Open-Source Online Test Platform," https://yaksh.fossee.in, 2025.

[3] "Comprehensive Human–Model Evaluation Across Three Logical Error Taxonomies," https://github.com/atmabodha/fossee-iitb, 2026.

[4] T. Phung et al., "Generating High-Precision Feedback for Programming Syntax Errors using Large Language Models," *arXiv preprint arXiv:2302.04662*, 2023.

[5] OpenAI, "OpenAI Models Overview," https://platform.openai.com/docs/models, 2025.

[6] Anthropic, "Claude Models Overview," https://docs.anthropic.com/en/docs/about-claude/models/overview, 2025.

[7] Google, "Gemini API Models," https://ai.google.dev/gemini-api/docs/models, 2025.

[8] DeepSeek AI, "DeepSeek-V3 Technical Report," https://github.com/deepseek-ai/DeepSeek-V3, 2025.

[9] Alibaba Cloud, "Qwen3 Technical Report," https://qwen.ai/blog?id=qwen3, 2025.

A Prompt Templates

A.1 Single-Label Prompt

```
<role>Expert Python Programming Assistant</role>

<task>
Identify the SINGLE most dominant logical error in the
provided Python code based on the taxonomy below.
</task>

<taxonomy>
  <category label="A">
    <title>Loop Condition</title>
    <description>Incorrect loops in for/while condition
    statements.</description>
  </category>
  <category label="B">
    <title>Condition Branch</title>
    <description>Incorrect expression in the if condition.
    </description>
  </category>
  <category label="C">
    <title>Statement Integrity</title>
    <description>Statement lacks a part of logical structure.
    </description>
  </category>
  <category label="D">
    <title>Output/Input Format</title>
    <description>Incorrect cin/cout or input/output statement.
    </description>
  </category>
  <category label="E">
    <title>Variable Initialization</title>
    <description>Incorrect declaration of variables.</
       description>
  </category>
```

```
  <category label="F">
    <title>Data Type</title>
    <description>Incorrect data type usage.</description>
  </category>
  <category label="G">
    <title>Computation</title>
    <description>Incorrect basic math symbols or operators.
    </description>
  </category>
  <category label="NONE">
    <title>No Error</title>
    <description>No logical error present.</description>
  </category>
</taxonomy>

<instructions>
1. Analyse the code inside the <code_to_analyze> tags.
2. If multiple errors exist, select the one that most
   fundamentally causes the code to fail.
</instructions>

<output_rules>
- Output ONLY the single label (A, B, C, D, E, F, G, or NONE).
- STRICT: Any text other than the label is a failure.
</output_rules>

<code_to_analyze>
{code_input}
</code_to_analyze>
```

Listing A.1: Full single-label prompt template.

A.2 Multi-Label Prompt

```
<role>Expert Python Programming Assistant</role>

<task>
Identify ALL applicable logical error types in the
provided Python code based on the taxonomy below.
</task>

<taxonomy>
```

```
  <!-- same taxonomy as single-label prompt above -->
</taxonomy>

<instructions>
1. Evaluate the code against every category in the taxonomy.
2. Select all labels that apply to the logic of the code.
</instructions>

<output_rules>
- Output labels as a comma-separated list (e.g., C,D,G).
- If NONE is chosen, it must be the ONLY label.
- Do NOT include spaces, titles, or descriptions.
- STRICT: No conversational filler or explanations.
</output_rules>

<code_to_analyze>
{code_input}
</code_to_analyze>
```

Listing A.2: Full multi-label prompt template.

B Raw Accuracy Data

For full reproducibility, Tables B.1 and B.2 present the per-run accuracy values.

Table B.1: Single-Label Raw Accuracy Per Run (%)

Model	**Run 1**		**Run 2**	
	Accuracy	**Consistency**	**Accuracy**	**Consistency**
GPT-5.2	69.5	85	69.5	85
GPT-OSS-120B	65.5	75	65.5	75
Claude Sonnet 4.5	57.5	66	57.5	66
Gemini 2.5 Flash	53.5	67	53.5	67
Qwen3-Coder	45.5	78	45.5	78
DeepSeek-V3.2	29.5	56	29.5	56

Table B.2: Multi-Label Partial Overlap Raw Data Per Run

	Run 1				**Run 2**			
Model	A	B	C	Jacc	A	B	C	Jacc
GPT-5.2	31.7	45.2	96.2	0.556	31.7	45.2	96.2	0.556
Claude Sonnet 4.5	32.7	38.5	93.3	0.511	32.7	38.5	93.3	0.511
GPT-OSS-120B	35.6	28.8	90.4	0.475	35.6	28.8	90.4	0.475
Gemini 2.5 Flash	28.8	10.6	73.1	0.325	28.8	10.6	73.1	0.325
DeepSeek-V3.2	13.5	26.9	70.2	0.304	13.5	26.9	70.2	0.304
Qwen3-Coder	7.7	24.0	76.0	0.291	7.7	24.0	76.0	0.291