FOSSEE Semester Long Internship Report

On

Benchmarking LLM for error detection for Yaksh

Submitted by

SHIVAM KUMAR

Vellore Institute of Technology Bhopal,Bhopal

Under the guidance of

Dr.Kushal Shah

and

Prof.Prabhu Ramachandran

FOSSEE, Indian Institute of Technology Bombay

23March,2026

Declaration

I hereby declare that this internship report, entitled " *Benchmarking LLM for error detection for Yaksh,*" is an original and authentic work submitted in partial fulfillment 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 : Shivam Kumar

Internship : FOSSEE Semester-Long Internship

Guide : Dr. Kushal Shah

Acknowledgments

I would like to take this opportunity to express my sincere gratitude to the FOSSEE team at IIT Bombay for granting me the privilege of participating in the Semester-Long Internship Program. This internship has been a valuable experience that helped me strengthen my technical knowledge and explore the intersection of open-source technology and sustainability.

I am deeply thankful to my mentor, **Dr. Kushal Shah,** for his guidance, thoughtful feedback, and continuous encouragement throughout the internship. His insights were instrumental in framing the research questions and shaping the direction of this project.

I would also like to acknowledge **Prof.Prabhu Ramachandran,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 would also like to thank my **team members** – Yuvraj Pandey, Harshit, Tanishi, and Priyanka for their collaborative efforts in the manual evaluation, taxonomy development, LLM API testing, and cross-verification studies.

I am grateful to my home institution, VIT BHOPAL,for their academic support throughout this period

Abstract

This report presents work carried out during a four-month internship at FOSSEE, IIT Bombay, focused on evaluating AI-generated pedagogical responses from the Yaksh coding platform,the PyPal dataset. The study addresses a central question: *can AI tutors reliably guide programming students without simply giving away answers?* We conducted an extensive exploratory data analysis (EDA) on 14,408 real-world Python coding interactions, revealing critical patterns including a dominant TypeError anomaly (92.9% attributable to a single system-level issue), low AI response quality (average semantic similarity of 0.216), and significant category imbalance in the dataset.

A manual pedagogical evaluation of 600 AI responses was performed by three evaluators using a five-dimension rubric (scale 0--4, maximum score 20), assessing conceptual accuracy, clarity, guidance quality, reasoning depth, and edge-case awareness.

A cross-verification study on 100 selected questions compared LLM-generated error classifications against human judgment across three independent taxonomy interpretations.

A review of three state-of-the-art benchmarks -- TutorBench, Dean of LLM Tutors, and GuideLM -- contextualizes our findings within the broader research landscape, along with a cost analysis for replicating these methodologies on the PyPal dataset.

Our findings confirm that while current AI models are competent at solving coding problems, they consistently fall short on the pedagogical dimension of teaching.

Keywords: AI tutoring, pedagogical evaluation, LLM benchmarking, error taxonomy, PyPal, Yaksh, FOSSEE

Contents

1. Introduction

1.1 Background

The Yaksh system is an AI-powered Coding tutoring assistant, developed and maintained by FOSSEE at IIT Bombay. Yaksh is widely used across Indian educational institutions for conducting Coding. When students submit incorrect code, PyPal generates natural-language explanations intended to guide them toward the correct solution without directly revealing it.

The effectiveness of such AI-generated feedback is not well understood. While large language models (LLMs) have demonstrated strong performance on code generation and debugging benchmarks such as HumanEval [7] and SWE-bench, the question of whether they can *teach* -- that is, guide students pedagogically rather than simply solving problems for them -- remains largely open.

1.2 Problem Statement

The central problem addressed in this work is threefold:

1. Data Quality: What are the characteristics, distributions, and anomalies present in the PyPal interaction dataset, and what do they reveal about student behaviour and AI response quality?
2. Pedagogical Effectiveness: How well do AI-generated responses score on established pedagogical criteria when evaluated by human experts?
3. Classification Reliability: Can LLMs reliably classify student errors using a structured taxonomy, and how do their classifications compare to human judgment?

1.3 Objectives

The specific objectives of this internship were:

- Conduct a comprehensive exploratory data analysis of the full 14,408-entry PyPal dataset.
- Participate in the manual pedagogical evaluation of 600 AI responses using a structured rubric.
- Perform a cross-verification study comparing LLM error classifications against human labels on a curated 100-question subset.
- Review state-of-the-art benchmarks for evaluating LLMs as pedagogical agents.
- Estimate the cost of replicating benchmark methodologies on the PyPal dataset.

1.4 Paper Organisation

The remainder of this report is organised as follows. Section 2 describes the PyPal dataset. Section 3 presents the exploratory data analysis. Section 4 covers the manual pedagogical evaluation of 600 AI responses. Section 5 describes the error taxonomy and LLM classification pipeline. Section 6 details the cross-verification study comparing human and LLM labels. Section 7 reviews related benchmarks and provides a cost analysis for replication. Section 8 concludes with a summary of findings and directions for future work.

---

2. Dataset Description

2.1 The PyPal Dataset

The primary dataset comprises 14,408 student-AI interaction records collected from the Yaksh platform at IIT Bombay. Each record captures a student's incorrect Python code submission and the corresponding AI-generated explanation.

Parameter	Value
Total Records	14,408
Language	Python
Source Platform	Yaksh, IIT Bombay
Concept Categories	6
Question Types	11
Key Columns	Question, Student Code, Error Type, AI Explanation, Concept, and others

2.2 The 100-Question Subset

A curated subset of 100 representative questions was later selected from the full dataset for the cross-verification study (Section 6). This subset was chosen because running multiple LLM APIs on all 14,408 entries would have been prohibitively expensive. The 100 questions were selected to be representative of the error distribution and concept coverage in the full dataset.

3. Exploratory Data Analysis

A comprehensive EDA was conducted on the full 14,408-entry dataset using Python (Pandas, NumPy, Matplotlib, Seaborn) in Google Colab. The analysis covered six dimensions: error type distribution, AI response quality, friendliness scoring, readability, error clustering, and category imbalance.

3.1 Error Type Distribution

Errors were classified into two broad categories: *logic-based failures* (where the code runs but produces incorrect output) and *code-crashing errors* (where the code raises a runtime exception).

Based on section 3.1 of the document, the data on Error Type Distribution is already presented in two tables.Error Class Distribution (Total Records: 14,408)

This table classifies errors into two broad categories:

Error Class	Count	Percentage
Logic Fails (incorrect output)	9,141	63.4%
Code-Crashing Errors (runtime exceptions)	5,267	36.6%
Total	**14,408**	**100%**

Breakdown of Code-Crashing Errors (Total: 5,267)

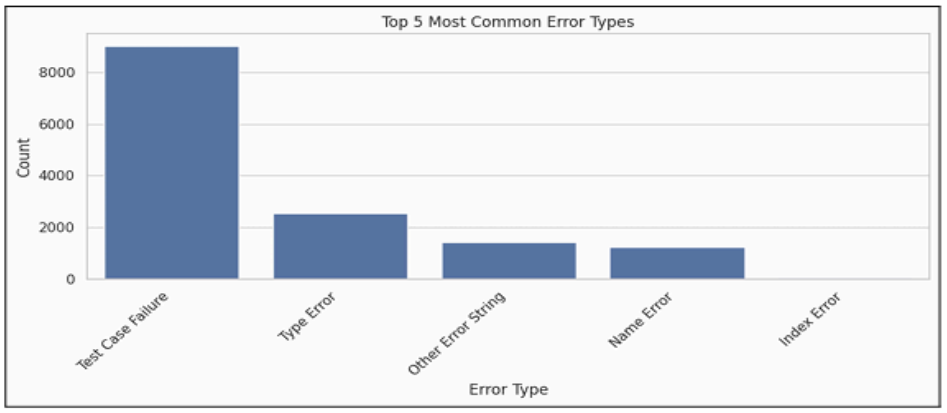
This table provides the breakdown of the 5,267 code-crashing errors by exception type:

Error Type	Count	% of Code-Crashing	Primary Root Cause
TypeError	2,544	48.3%	Incorrect parameter handling, type mismatches
NameError	1,247	23.7%	Undefined variables, typos in variable names

ZeroDivisionError	607	11.5%	Missing edge-case checks for division
UnboundLocalError	390	7.4%	Variable referenced before assignment
AttributeError	210	4.0%	Accessing non-existent attributes
EOFError	88	1.7%	Input handling issues
IndexError	81	1.5%	List/sequence index out of range
ValueError	70	1.3%	Invalid value for a given operation
KeyError	16	0.3%	Dictionary key access errors
Others	14	0.3%	RecursionError, OverflowError, etc.

However, when viewed by overall frequency (including logic-based "Test Case Failures"), the distribution changes significantly. Test Case Failure is the single most common error type in the dataset, occurring over 8,000 times -- more than three times as frequent as the next most common type (TypeError at ~2,500).

- **Most Common Error Types:** This bar chart displays the most frequent types of student errors, showing that Test Case Failure is the most common issue, occurring over 8,000 times.



- **AI Interpretation by Error Type:** This bar chart shows the Average Semantic Match (relevance) of the AI's feedback for different error types.

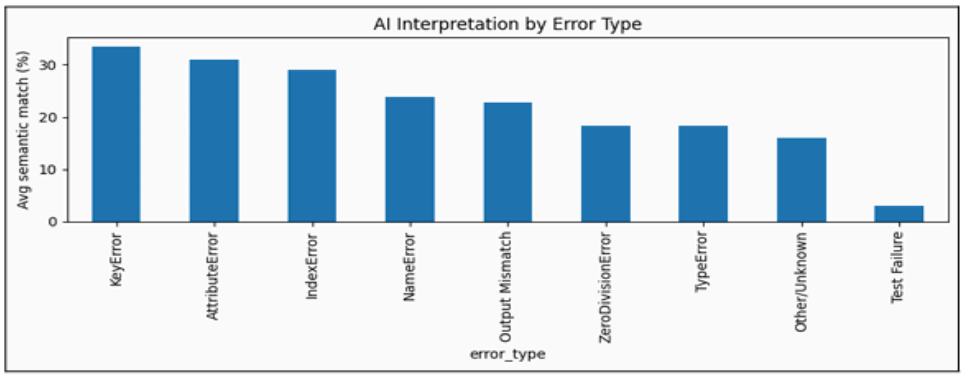


Figure 1: Top 5 Most Common Error Types in the PyPal dataset and AI Interpretation by Error Type.

3.2 TypeError Anomaly

A striking finding was that 92.9% of all TypeErrors in the dataset correspond to the sub-category "Function Argument Mismatch". Drilling further into this sub-category reveals that 100% of these cases produce the identical message: "main() takes 0 positional arguments but 1 was given". This is not a genuine student mistake but a system-level issue in the Yaksh platform's function invocation mechanism.

TypeError Sub-Category	Percentage	Count (approx.)
Function Argument Mismatch	92.9%	~2,363
String to Int Conversion	~0.9%	~23

Int to String Conversion	~0.7%	~18
Other Type Error	~5.5%	~140

Note: The document highlights that the Function Argument Mismatch is a system-level issue, as 100% of these cases produce the identical error message: "main() takes 0 positional arguments but 1 was given".

- **Type Error Sub-categories Distribution:** This chart shows the percentage breakdown of different types of `TypeError`. It indicates that "Function Argument Mismatch" is the overwhelming cause, accounting for **92.9%** of all `TypeError` sub-categories.

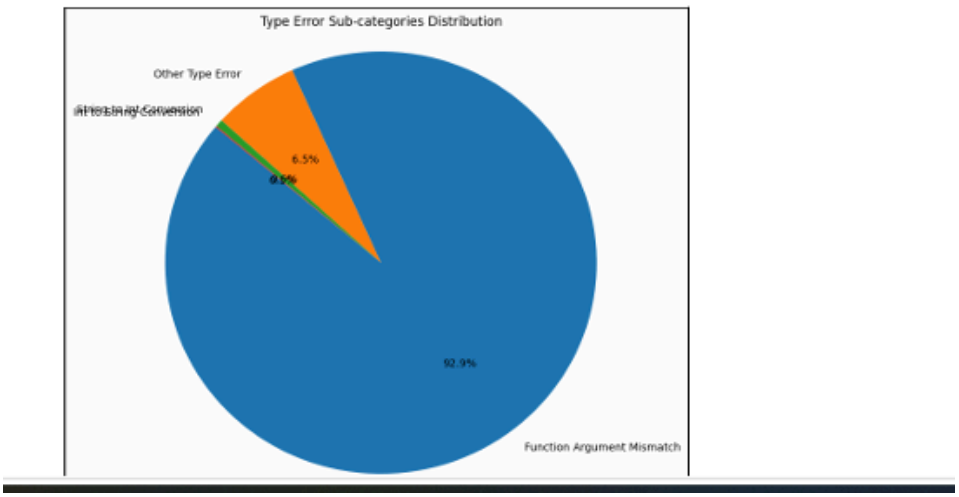


Figure 2: TypeError Sub-categories Distribution (pie chart) and Distribution of Type Error Sub-categories (bar chart).

This anomaly inflates the apparent error rate and should be addressed at the platform level rather than through AI tutoring.

3.3 AI Response Quality (Semantic Similarity)

The semantic similarity between AI-generated explanations and reference solutions was computed to assess how well the AI understood each error.

Metric	Value
Average Semantic Similarity Score	0.216 (on a 0–1 scale)

Correct Interpretation Rate	~40%
Test Failure Category Match	2.91% (lowest of all categories)

With average semantic similarity of 0.216, the automatically generated explanations often fail to accurately address student code errors, showing significant variability by error type.The "Test Failure" category performed worst with a match rate of only 2.91%, suggesting a significant blind spot. Notably, even the best-performing category (KeyError at 33.52%) achieves only a third semantic alignment -- confirming that low response quality is systemic, not confined to a single error type.

A scatter plot of response length versus semantic match further confirms that longer responses do not correlate with better quality. The bulk of responses cluster between 50--200 words with semantic match scores between 0.05--0.45, and no positive trend is visible.

Error Type	Avg. Semantic Match	% Containing Actionable Advice	% Containing Code Snippets	Avg. Response Length (Words)
KeyError	33.52%	75.0%	87.5%	114.8
AttributeError	31.06%	82.4%	69.5%	114.3
IndexError	29.10%	82.7%	67.9%	106.1
NameError	23.78%	89.7%	68.4%	113.2
Output Mismatch	22.70%	86.7%	60.2%	111.1
ZeroDivisionError	18.36%	84.6%	56.6%	115.6
TypeError	18.33%	86.7%	69.9%	113.9
Other/Unknown	16.07%	88.1%	74.7%	128.0
Test Failure	2.91%	76.8%	46.4%	96.9

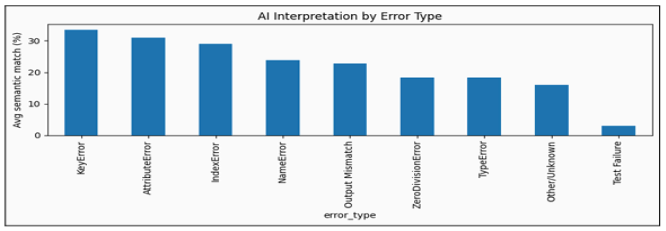


Figure 3 AI Interpretation by Error Type (table and bar chart).

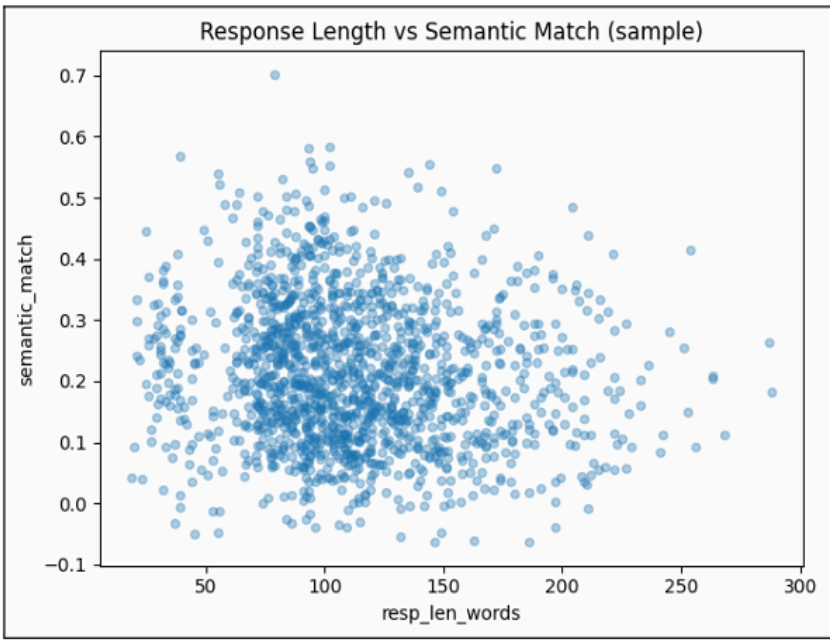


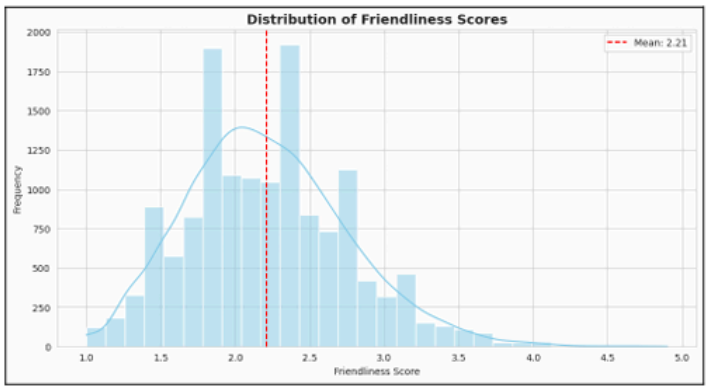
Figure 4: Response Length vs Semantic Match (scatter plot).

3.4 Friendliness Scoring

AI responses were scored for tone and approachability on a scale of 1 (Very Unfriendly) to 5 (Very Friendly).

Rating	Percentage
Very Unfriendly	34.0%
Neutral	58.3%
Friendly	7.6%
Very Friendly	0.2%

The average friendliness score of 2.21 out of 5 indicates that the majority of AI responses are either neutral or actively unfriendly in tone. Over a third (34%) were rated "Very Unfriendly", while only 7.8% were rated Friendly or above. The histogram of scores shows a roughly normal distribution centred around the mean of 2.21, with a long tail toward higher friendliness scores that very few responses reach. For a platform serving beginner programmers, this is a significant concern, as discouraging feedback can negatively impact student motivation and learning outcomes.



This chart above shows the frequency of all AI friendliness scores, revealing that mo responses cluster around the mean of 2.21, which is marked by a red dashed line.

Figure 5: Distribution of Friendliness Scores (histogram with mean line at 2.21).

3.5 Readability Analysis

Flesch-Kincaid readability analysis was applied to assess whether AI explanations are pitched at an appropriate level for the target audience (beginner Python programmers).

Readability Level	Percentage
Easy (accessible to beginners)	28%
Medium (general audience)	68%
Hard (advanced vocabulary/structure)	3.4%

Readability Distribution by Concept

Concept 6 has the highest proportion of "Easy" responses (41.93%), while Concept 1 has the lowest (21.14%) coupled with the highest "Hard" percentage (5.25%). Given that the target audience consists of introductory programming students, a higher proportion of easy-to-read responses would be desirable -- particularly for Concept 1, which is also the most frequently occurring concept in the dataset.

Concept	Easy %	Hard %	Medium %
Concept 1	21.14	5.25	73.60

Concept 2	29.38	3.02	67.60
Concept 3	37.55	2.05	60.40
Concept 4	24.83	2.24	72.93
Concept 5	27.32	3.66	69.02
Concept 6	41.93	1.50	56.57

3.6 Error Clustering

Analysis of error co-occurrence patterns among students revealed common combinations:

Based on your request, here is the data from section 3.6 of the document, "Error Clustering," presented in a table format:

Error Combination	Number of Students
NameError + TypeError (top pair)	275
NameError + TypeError + ZeroDivisionError (top triplet)	153

The document notes that these clusters suggest that students who make one type of fundamental mistake often make related mistakes as well, indicating common underlying misconceptions (such as confusion about variable scope and type handling).

3.7 Category Distribution and Imbalance

The dataset exhibits significant imbalance across concept categories. The full distribution is as follows:

The dataset's imbalance heavily features list and basic-operation errors, which may lead to AI models performing poorly on less common categories like conditional logic and unit conversion. Furthermore, Data Structures (Dictionary and List) problems generally have the longest student submissions (~350–450 characters), indicating higher complexity, whereas Unit Conversion and Conditional Logic problems are shorter and simpler.

As shown in the distribution chart, Data Structures – List and Basic Operations dominate the problem set, together accounting for over half of all available exercises. Meanwhile, critical foundational areas such as Conditional Logic and Unit Conversion have relatively few problems. This uneven spread suggests students spend disproportionate time in certain categories, likely contributing to the stagnation patterns observed in earlier sections.

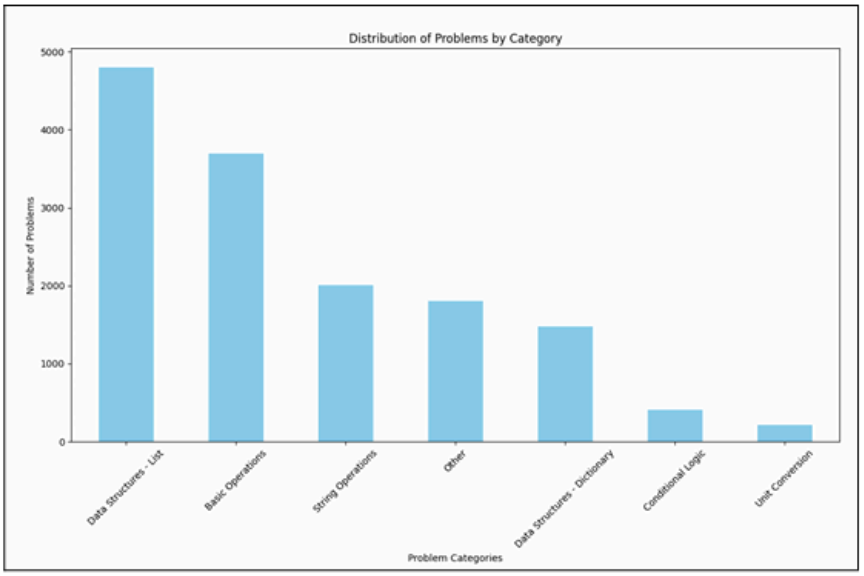


Figure 7: Distribution of Problems by Category (bar chart).

---

4. Manual Pedagogical Evaluation

4.1 Methodology

A team of three evaluators (Me, Yuvraj Pandey, and Tanishi) independently assessed approximately 600 AI-generated responses (roughly 200 per evaluator). Each response was rated on a five-dimension rubric:

Dimension	Description	Scale
Conceptual Accuracy	Is the explanation factually correct about the underlying concept or algorithm?	0–4
Clarity and Structure	Is the explanation clearly written and logically organised?	0–4

Helpful Guidance / Hint Quality	Does the response teach *how to think* about the problem without giving the solution?	0–4
Reasoning Depth and Intuition	Does the response explain *why* the approach works, not just *what* to do?	0–4
Edge Cases / Constraints Awareness	Does the response mention important pitfalls, boundary conditions, or constraints?	0–4

Total Score Interpretation Scale

The maximum possible score was 20 (5 dimensions x 4 points each). The scores were interpreted as follows:

Total Score	Interpretation
18–20	Excellent teaching explanation
14–17	Good
10–13	Adequate
5–9	Weak
0–4	Poor or unusable

4.2 Observations

The manual evaluation of 600 AI responses revealed that while many responses are factually accurate, they frequently fall short on pedagogical dimensions -- particularly on *Helpful Guidance / Hint Quality* and *Reasoning Depth*. A common pattern was responses that correctly identified the error but then provided the solution directly, rather than guiding the student to discover it independently.

This pattern mirrors the "over-helpfulness" problem identified by GuideLM [3], where base models (without pedagogical fine-tuning) tend to act as solvers rather than tutors.

---

5. Error Taxonomy and LLM Classification

5.1 Taxonomy Development

The team developed a Pedagogical-Granular (PG) error taxonomy [4] with 10 categories (A through J) plus a "None" (N) category for correctly handled submissions. This taxonomy was primarily designed by Harshit, Tanishi, and Yuvraj and was used to classify the types of logical and structural errors in student submissions.

Yuvraj's Error Taxonomy:

Code	Error Category	Description
A	Output/Input Format Errors	Incorrect formatting of input reading or output printing
B	Computation/Calculation Errors	Wrong arithmetic, off-by-one, or formula mistakes
C	Logic/Flow Control Errors	Incorrect conditional branching or control flow
D	Data Type Handling Errors	Type mismatches, incorrect casting, or type assumptions
E	Variable/Naming Errors	Undefined variables, scope issues, or naming mistakes
F	Function Definition/Parameter Errors	Wrong function signatures, missing parameters, or return issues
G	Loop/Iteration Errors	Infinite loops, wrong range, or iteration logic errors
H	Edge Case / Boundary Errors	Missing handling for empty inputs, zero values, or boundary conditions
I	Syntax Errors	Malformed Python syntax (indentation, brackets, colons, etc.)
J	Miscellaneous / Other	Errors not fitting any of the above categories
N	None	No error detected; the submission was correct or properly handled

Tanishi's error Taxonomy:

Type of Logic error	Description
Loop condition	Incorrect loops in the for/while condition
Condition branch	Incorrect expression in the if condition
Statement integrity	Statement lacks a part of logical structure
Output/Input format	Incorrect cin/cout statement
Variable initialization	Incorrect declaration of variables
Data type	Incorrect data type
Computation	Incorrect basic math symbols

Harshit's Error Taxonomy:

This taxonomy serves as the single-label classification standard for all models, annotations, and analysis.

Code	Category Name	Definition (Primary Failure)
A	↔ Boundary & Indexing	Loop bounds or index space misalignment (off-by-one, skipped element, extra iteration).
B	Conditional / Boolean	Wrong condition, incorrect boolean operator, reversed logic, unreachable branch.
C	State Management	Incorrect variable lifecycle (not resetting counters, stale values, wrong update order).
D	Algorithmic Strategy	Incorrect approach or data structure; logic fundamentally does not solve the problem.
E	Edge Case Handling	Failure only on boundary inputs (empty, single element, identical values, zero/negative).
F	Specification	Misunderstanding problem statement or output contract.

General Rules

- Exactly ONE category per code snippet.
- Choose the dominant root cause.
- Do not classify symptoms.

5.2 LLM API Testing

Following taxonomy development, the team ran classification tasks through multiple LLMs -- both open-source and closed-source -- to test how reliably these models could categorise errors using the A--J taxonomy. Models tested included GPT-5.2, Claude Sonnet 4.5, Gemini 2.5 Flash, DeepSeek-v3.2, Qwen3-Coder, and GPT-OSS-120b.

Each LLM was given questions from the dataset and asked to identify error categories based on the A--J taxonomy. LLM responses were collected and compared against the taxonomy definitions. F1 scores and other classification metrics were calculated, and reports were generated to compare how different LLMs performed on the error classification task.

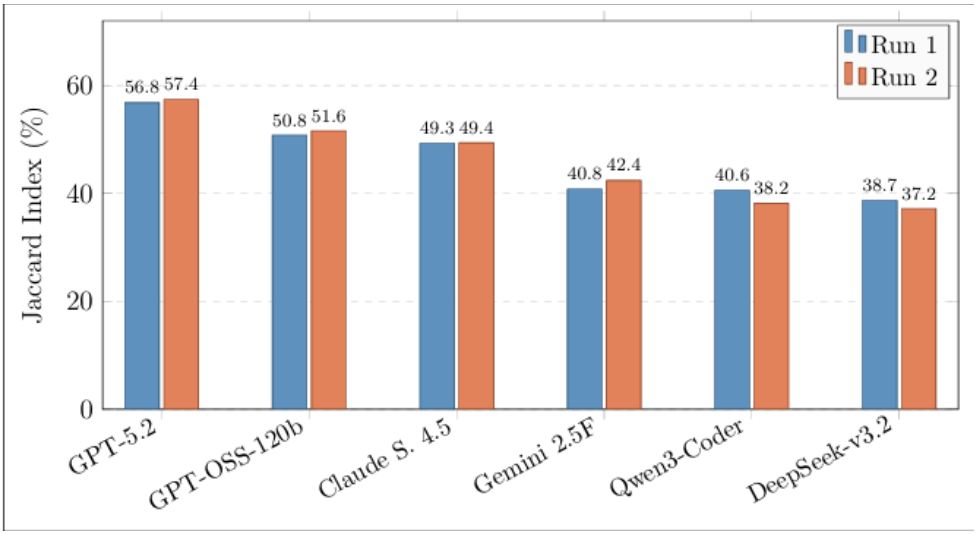


Figure 1: Multi-Label Jaccard Index scores per model for Run 1 and Run 2 performed by Yuvraj

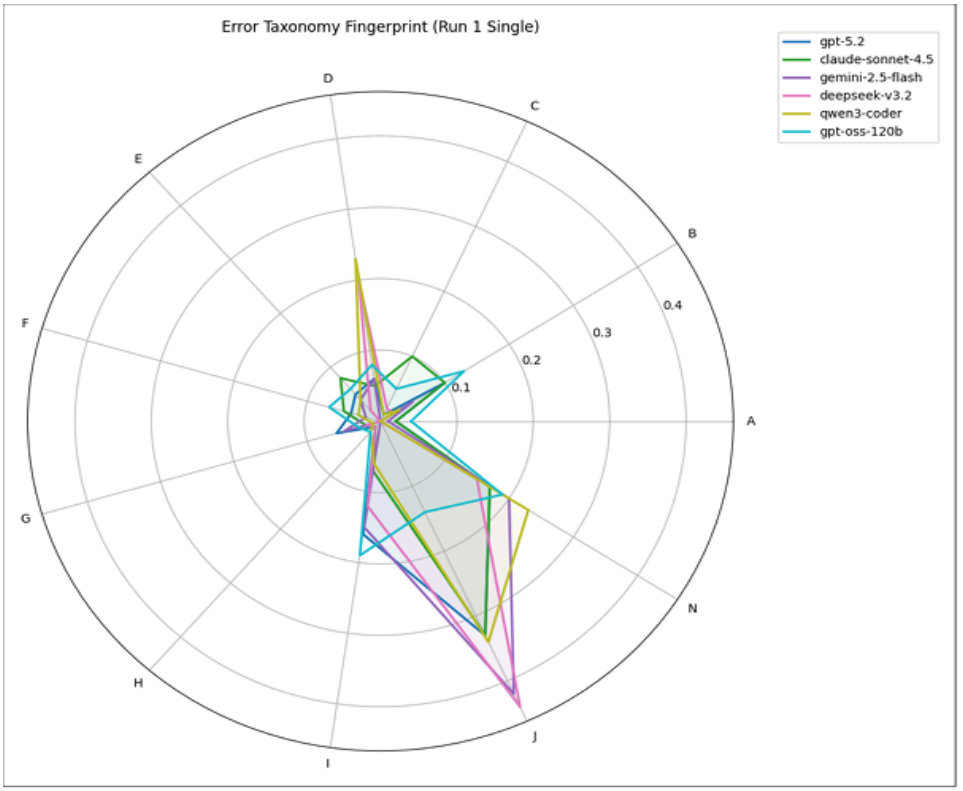


Figure 2 : Error Taxonomy Fingerprint in Single Label mode (Run 1) performed by Yuvraj

---

6. Cross-Verification Study: Human vs. LLM Classification

6.1 Objective

After the LLM API classification was complete, the team needed to validate LLM outputs against human judgment. This cross-verification study was conducted by me and Priyanka.

6.2 Procedure

1. A subset of 100 questions was selected from the 14,408-entry dataset. The selection was driven by cost constraints: running multiple paid LLM APIs on the full dataset was prohibitively expensive.
2. Then me and Priyanka independently reviewed each of the 100 questions, reading the student's code, the error, and the AI-generated explanation.
3. For each question, human labels were assigned three times, each time following the taxonomy interpretation of a different team member (Harshit's interpretation, Tanishi's interpretation, and Yuvraj's interpretation). This yielded 300 total human classifications per annotator (100 questions x 3 taxonomy styles).
4. Each question could receive multiple category labels (e.g., "A,D" or "J,I"), as student errors often span more than one category.
5. Questions with no identifiable error were marked as "None" (N).

The rationale for applying three taxonomy interpretations was to assess inter-annotator consistency: since the A--J taxonomy was developed collaboratively, each team member had a slightly different interpretation of category boundaries. By classifying the same 100 questions under each interpretation, the study reveals where the taxonomy is unambiguous and where it admits disagreement.

6.3 Evaluation Metrics

Agreement between human and LLM classifications was measured using four metrics:

Metric	Definition
Case A -- Precision	Proportion of LLM-predicted categories that match human labels
Case B -- Recall	Proportion of human-labeled categories captured by LLM predictions
Case C -- Any-Overlap	Whether there is *any* intersection between human and LLM label sets
Case D -- Jaccard Index	Intersection over union of the human and LLM label sets

6.4 Findings

The 100-question cross-verification study yielded the following observations:

- 19 out of 100 questions were classified as "None" (N) -- i.e., the AI response was correct or the submission had no identifiable error. This means that roughly 81% of the selected questions contained errors that required classification.
- Inter-taxonomy disagreement was observed across the three interpretation styles. The same question would sometimes receive different category labels depending on whether Harshit's, Tanishi's, or Yuvraj's taxonomy interpretation was applied. This highlights inherent ambiguity in the A--J taxonomy boundaries.
- Questions frequently received multiple category labels (e.g., a single student error could be classified as both "A" and "D"), reflecting the multi-faceted nature of programming errors.

These disagreements are not a weakness but an informative finding: they reveal where the taxonomy needs refinement and where category definitions overlap.

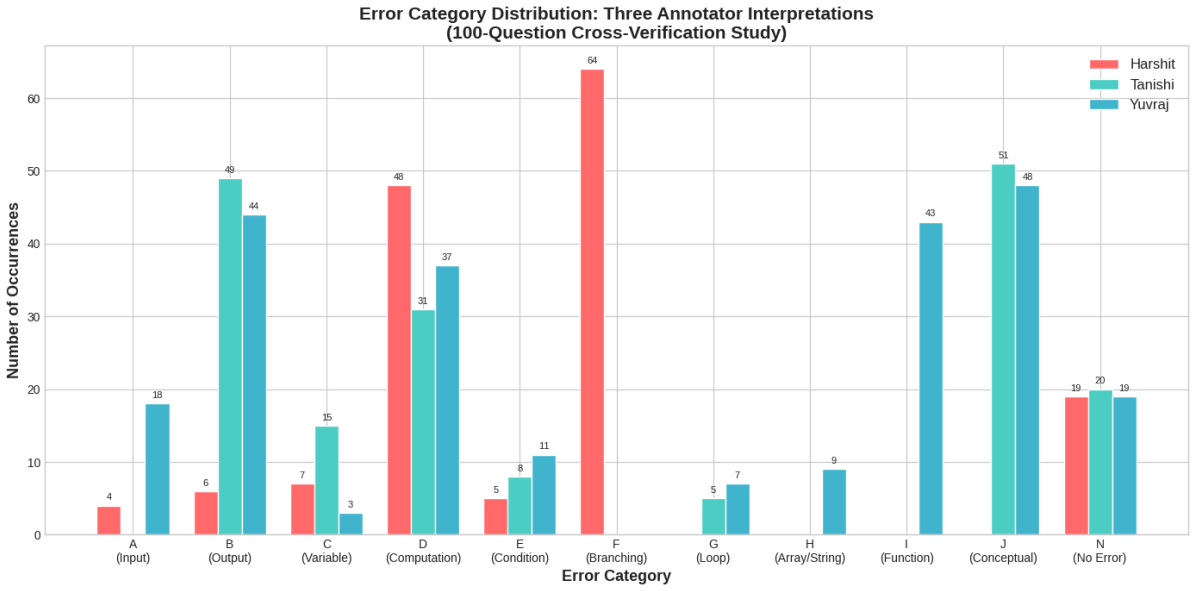


Fig 1: error categories distribution

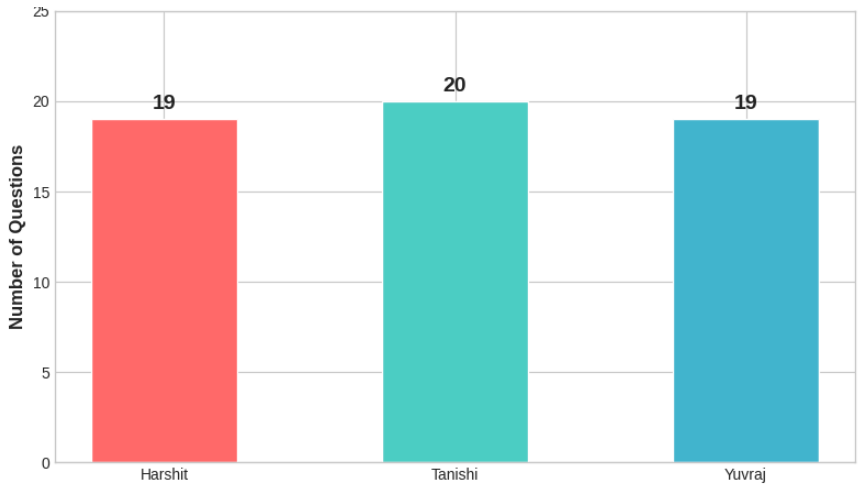


Fig: Questions Classified as N

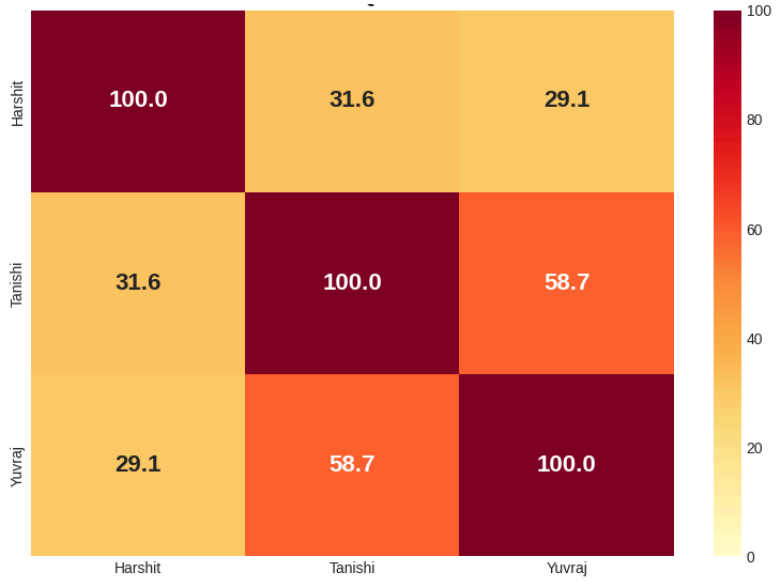


Fig: Average Inter-Annotator Agreement(Avg Jaccard Index %)

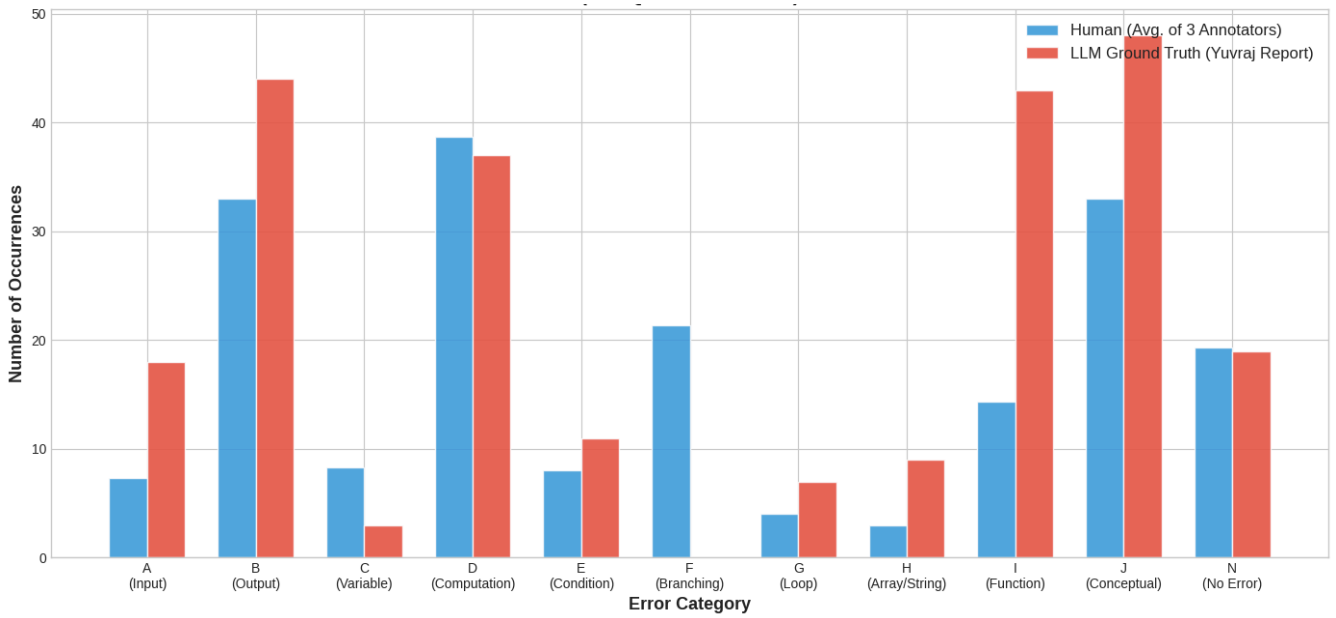


Fig: Human Classification vs LLM

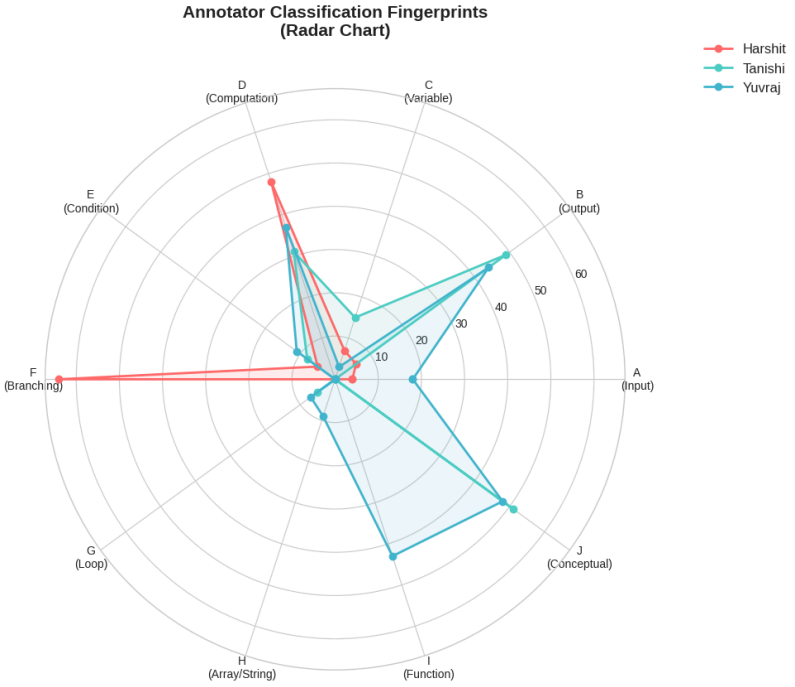


Fig: Questions Classified as N

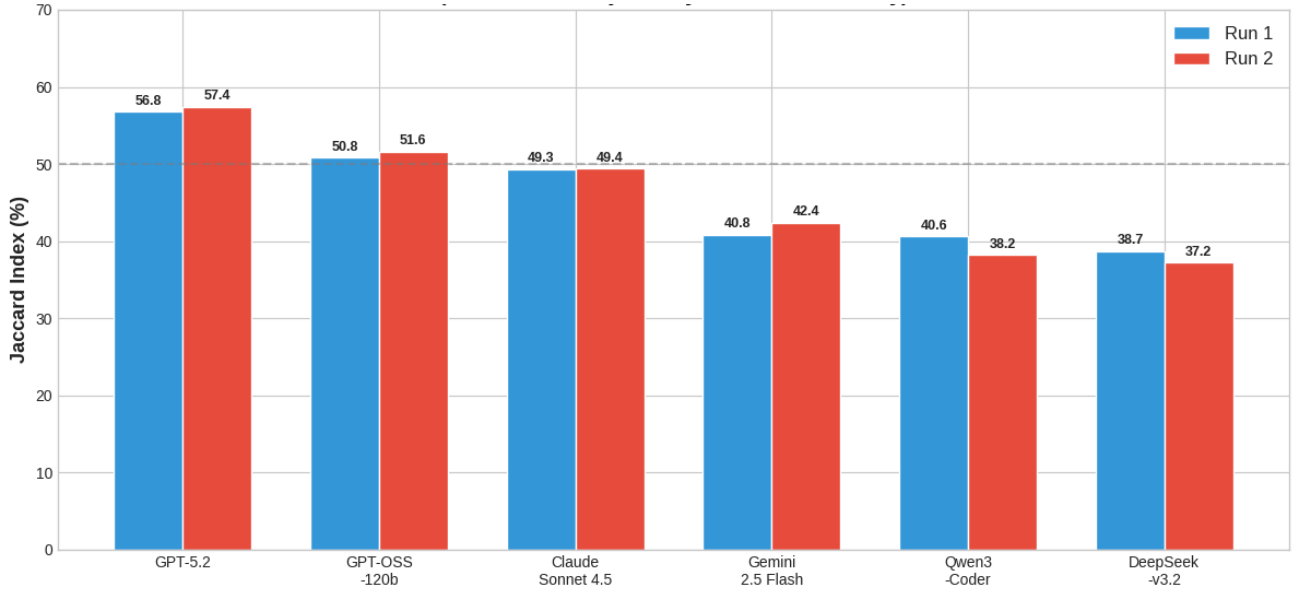


Fig: LLM Multi-Label Accuracy:Jaccard Index per Model

---

7. Benchmark Literature Review and Cost Analysis

7.1 Overview

While the empirical work described in Sections 3--6 was ongoing, the team also researched how the academic community evaluates AI tutors. This was initially a collaborative effort; after the initial research, I continued to carry forward the benchmarking task and its cost of replication analysis. Three state-of-the-art benchmarks were studied in depth.

7.2 TutorBench

TutorBench [1] is a benchmark developed by Scale AI comprising 1,490 expert-level tutoring conversations across six STEM disciplines. It evaluates LLMs on three pedagogical pillars: *adaptive explanation generation*, *assessment and feedback*, and *active learning support*. A rubric-based scoring system (with a critical --5 penalty for revealing answers) is used, with an LLM judge (Claude Sonnet 4) evaluating responses. The best-performing model, Gemini 2.5 Pro, achieved only 55.7% overall -- far below the human-level threshold typically set at 85% or above. Key failure modes include "lecture mode" bias (long, unsolicited explanations), visual hallucination in multimodal tasks, and a consistent empathy gap.

Top Performers

Rank	Model	Text-Only %	Multimodal %	Overall %	CI
1	**Gemini 2.5 Pro**	57.05	54.53	**55.65**	1.11
2	**GPT 5**	57.03	53.97	**55.33**	1.02
3	o3 Pro	56.07	53.45	54.62	1.02
4	o3 Medium Effort	54.11	51.68	52.76	1.00
5	o3 High Effort	52.91	51.43	52.09	1.01
6	Claude Opus 4.1 Thinking)	51.65	50.08	50.78	1.05
7	Claude Opus 4 Thinking)	50.40	49.14	49.71	1.02
8	Claude Opus 4.1	49.51	45.72	47.40	1.06
9	Claude 3.7 Sonnet Thinking)	45.67	47.07	46.45	1.03
10	Claude Opus 4	47.79	43.59	45.46	1.06
11	Llama 4 Maveric k	39.54	40.73	40.20	1.00
12	GPT 4o	39.10	33.74	36.12	0.96

Model	Text-Only %	CI
gpt-oss-120b	56.01	1.49
gpt-oss-20b	49.01	1.53
DeepSeek-R1	48.38	1.50

7.3 Dean of LLM Tutors

The Dean of LLM Tutors framework [2], developed at Monash University, introduces a governance architecture where a supervisor model (the "Dean") audits a tutor model's output across 16 dimensions spanning feedback content, effectiveness, and hallucination detection. The study evaluated 2,000 feedback instances across 85 university-level CS assignments. After fine-tuning, GPT-4.1 achieved 79.8% overall accuracy as a feedback evaluator -- reaching human expert parity (78.3%). This result is significant because it suggests that automated quality assurance of AI tutoring is now feasible at scale.

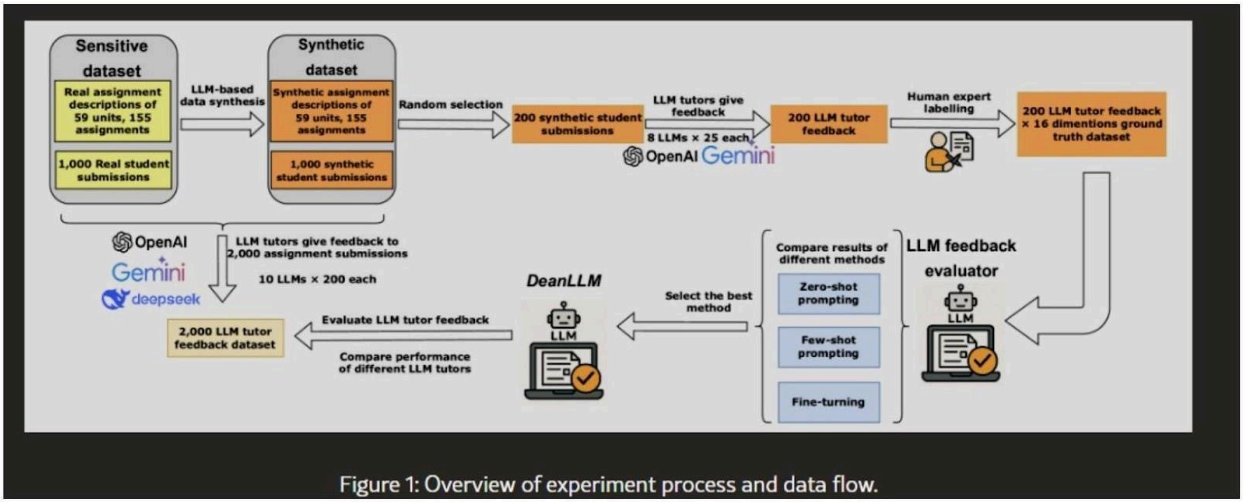


Figure 1: Overview of experiment process and data flow.

7.4 Evaluation Framework: 16 Dimensions

The core contribution of this paper is the 16 Dimension Rubric used to audit AI Tutors, split into three critical categories.

DeanLLM Evaluator Performance

LLM Feedback Evaluator	Overall Accuracy	Overall F1	Content Accuracy	Effectiveness Accuracy	Hallucination Accuracy
Zero-shot Prompting					
GPT 4.1	0.734	0.738	0.607	0.811	0.807
o4-mini	0.738	0.742	0.652	0.780	0.813
o3	0.741	0.742	0.603	0.816	0.833
o3-pro	**0.744**	**0.745**	0.607	**0.821**	**0.837**
Few-shot Prompting					
GPT 4.1	0.733	0.737	0.598	0.816	0.810

LLM Feedback Evaluator	Overall Accuracy	Overall F1	Content Accuracy	Effectiveness Accuracy	Hallucination Accuracy
o4-mini	**0.749**	**0.752**	**0.657**	0.807	0.800
o3	0.735	0.736	0.595	0.811	0.837
o3-pro	0.743	0.744	0.590	0.824	0.860
Fine-tuned					

GPT 4.1 (plain-labelled)	**0.798**	**0.794**	**0.737**	**0.843**	0.813
Human Baseline					
Human Average 3 coders)	0.783	0.826	0.659	0.813	0.963

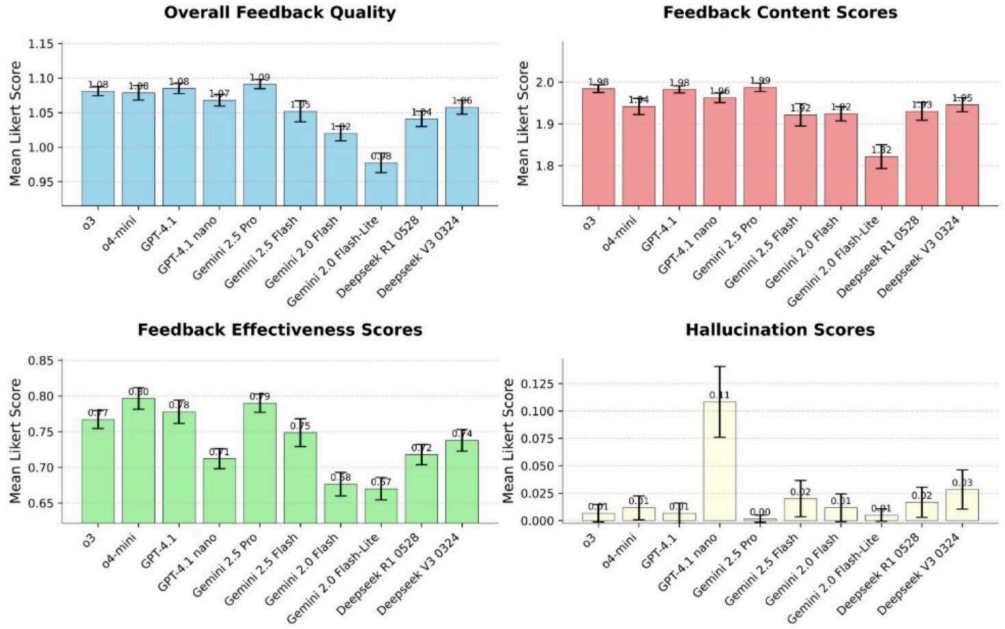


Figure : Feedback quality scores of different LLM Tutors

7.4 GuideLM

GuideLM [3], from UNSW and Princeton, takes a different approach: supervised fine-tuning (SFT) of GPT-4o and GPT-4o-mini on 528 carefully curated Socratic tutoring pairs drawn from an initial pool of 129,000 forum posts. The fine-tuned models (GuideLM and GuideLM-mini) showed a +25.4% improvement in Socratic guidance and approximately 58% improvement in conciseness, while reducing over-helpfulness by up to 31.7%. However, this came at a cost of 9--20% reduction in conceptual accuracy -- confirming a fundamental trade-off between teaching ability and raw c Problem solving.

7.5 Connection to Our Work

Our Work	Closest Benchmark	Similarity
5-Dimension Manual Rubric (Section 4)	GuideLM (9-Point Rubric)	Both use human experts to rate AI responses on pedagogical quality
Error Taxonomy A--J (Section 5)	Dean of LLM Tutors (16 Dimensions)	Both classify feedback quality and check for errors and hallucinations
Human vs. LLM Cross-Verification (Section 6)	TutorBench (LLM Judge)	Both compare LLM outputs against human ground truth

All three benchmarks reveal a common insight: teaching ability and raw accuracy are inversely correlated. The more an LLM is optimised to teach (guide, hint, question), the less accurate it becomes at solving problems -- and vice versa. This tension is central to the findings of the present study.

7.6 Positioning Within the Benchmark Landscape

Parameter	TutorBench	Dean of LLM Tutors	GuideLM	This Work
Type	Benchmark	Evaluation Framework	Fine-tuned Model	Dataset Analysis + Human Evaluation
Dataset Size	1,490 conversations	2,000 feedback instances	528 curated pairs	14,408 interactions
Domain	6 STEM subjects	43 CS courses	CS1 (C/C++)	Python (Yaksh/IIT Bombay)
Evaluation	LLM judge	16-dim rubric (LLM + human)	9-point rubric (human)	5-dim rubric (human) + cross-verification
Key Finding	Best model at 55.7%	Fine-tuned GPT-4.1 reaches human parity	+25.4% Socratic, --9--20% accuracy	Semantic match 0.216; friendliness 2.21/5

Our work is complementary to these benchmarks. While TutorBench, Dean, and GuideLM operate primarily on curated or synthetic datasets, our analysis is grounded in real-world, large-scale student interaction data from an Indian educational institution, providing ecological validity that synthetic benchmarks lack.

7.7 Cost of Replication

An important practical consideration for future work is the cost of replicating established benchmark methodologies on the PyPal dataset. The following estimates were computed based on current API pricing.

Benchmark Approach	Low-Cost Estimate (GPT-4o-mini)	High-Cost Estimate (GPT-4o)
TutorBench-style evaluation (14K dataset, 3 models)	~INR 3,000	~INR 20,000
GuideLM-style fine-tuning (14K dataset)	~INR 1,300	~INR 21,500
Combined Total	**~INR 4,300**	**~INR 41,500**

Note: Only GuideLM's official project cost (USD 250) is stated in the original paper [3]. All other figures above are our estimates based on OpenAI API pricing. The TutorBench [1] and Dean of LLM Tutors [2] papers do not disclose costs.

The cost disparity between low and high estimates (roughly 10x) highlights that model selection is a critical economic decision. For resource-constrained projects, GPT-4o-mini offers a viable path to replication at approximately INR 4,300 total.

A detailed cost analysis with per-step breakdowns for each benchmark methodology is available in the companion document: *Benchmark Analysis: Evaluating LLMs as Pedagogical Agents*

---

8. Conclusion and Future Work

8.1 Summary of Findings

This study conducted a multi-faceted evaluation of AI-generated pedagogical responses om the Yaksh platform, using the PyPal dataset. The key findings are:

1. **The PyPal dataset reveals significant anomalies.** The TypeError anomaly (92.9% from a single system-level message) inflates error statistics and should be addressed at the platform level. Category imbalance (Data Structures--List dominating at ~4,800 entries while Unit Conversion has only ~200) limits the representativeness of the dataset.
2. **AI response quality is low.** An average semantic similarity of 0.216 and a correct interpretation rate of only ~40% indicate that the AI-generated explanations in the PyPal dataset frequently fail to address the student's actual error. The Test Failure category is particularly problematic, with a match rate of just 2.91%.
3. **AI responses are not student-friendly.** With an average friendliness score of 2.21/5 and 34% of responses rated "Very Unfriendly", the current system's tone is inappropriate for its target audience of beginner programmers.
4. **Human evaluation confirms the solver-vs-tutor gap.** The manual evaluation of 600 responses found a consistent pattern of responses that correctly identify errors but provide direct solutions rather than pedagogical guidance -- the same "over-helpfulness" problem documented by GuideLM [3].
5. **Taxonomy boundaries are ambiguous.** The cross-verification study showed inter-annotator disagreement when the same questions were classified under different team members' taxonomy interpretations, indicating that the A--J taxonomy requires further refinement.
6. **Benchmark replication is economically feasible.** At approximately INR 4,300 using

GPT-4o-mini, replicating TutorBench and GuideLM methodologies on the PyPal dataset is within reach for academic research budgets.

8.2 Recommendations for Future Work

Based on the findings of this study, the following directions are recommended:

- **Address the TypeError anomaly** at the Yaksh platform level, as it is a system configuration issue rather than a student error.
- **Apply GuideLM's fine-tuning approach to the PyPal dataset:** curate the top ~500 "Socratic" examples from the 14,408 entries and fine-tune a model specifically for pedagogical guidance in the Python/Yaksh context.
- **Deploy a "Dean" evaluator model** (following the Dean of LLM Tutors framework) to automatically audit AI tutor quality before responses reach students.
- **Rebalance the dataset** by collecting or augmenting data for under-represented categories (Conditional Logic, Unit Conversion).
- **Refine the A--J taxonomy** to reduce ambiguity at category boundaries, informed by the inter-annotator disagreement patterns observed in the cross-verification study.
- **Extend testing to open-source models** (Llama, DeepSeek) for cost-effective deployment at scale.
- **Improve the friendliness and tone** of AI responses, potentially through fine-tuning on positively-toned examples or adding tone-control prompts.

References

[1] R. S. Srinivasa et al., "TutorBench: A Benchmark To Assess Tutoring Capabilities Of Large Language Models," *arXiv preprint arXiv:2510.02663*, October 2025. Available: https://arxiv.org/abs/2510.02663

[2] K. Qian, Y. Cheng, R. Guan, W. Dai, F. Jin, K. Yang, S. Nawaz, Z. Swiecki, G. Chen, L. Yan, and D. Gasevic, "Dean of LLM Tutors: Exploring Comprehensive and Automated Evaluation of LLM-generated Educational Feedback via LLM Feedback Evaluators," *arXiv preprint arXiv:2508.05952*, August 2025. Available: https://arxiv.org/abs/2508.05952

[3] E. Ross, Y. Kansal, J. Renzella, A. Vassar, and A. Taylor, "Supervised Fine-Tuning LLMs to Behave as Pedagogical Agents in Programming Education," *arXiv preprint arXiv:2502.20527*, February 2025. Available: https://arxiv.org/abs/2502.20527

[4] S. Lee et al., "A Pedagogical-Granular Taxonomy for Classifying Student Programming Errors,"

arXiv preprint arXiv:2404.19336, April 2024. Available: https://arxiv.org/abs/2404.19336

[5] J. Hattie and H. Timperley, "The Power of Feedback," *Review of Educational Research*, vol. 77, no. 1, pp. 81--112, 2007.

[6] FOSSEE Project, IIT Bombay. Available: https://fossee.in

[7] M. Chen et al., "Evaluating Large Language Models Trained on Code," *arXiv preprint arXiv:2107.03374*, July 2021. Available: https://arxiv.org/abs/2107.03374