VIDYALANKAR INSTITUTE OF TECHNOLOGY

Mumbai, India

Semester Long Internship — Spring 2026

Duration: 1 February 2026 – 30 June 2026

Project Report

on

Evaluation and Enhancement of an LLM-Based Error Classification Pipeline, and a Persona-Driven AI Tutor for Python Programming Education

Bug Fixes and Hypothesis Testing on Model Behavior, and the H.C. Verma Tutoring Persona for FOSSEE / YAKSH

Submitted by: Manini Jaiswal

Vidyalankar Institute of Technology, Mumbai, India

Under the guidance of:

Dr. Kushal Shah

Prof. Prabhu Ramachandran

June 30, 2026

Acknowledgements

I would like to express my sincere gratitude to the FOSSEE team at IIT Bombay for the opportunity to spend this internship working on AI-assisted programming education. I am especially thankful to my supervisor, Dr. Kushal Shah, and to Prof. Prabhu Ramachandran, for their patient guidance, their willingness to question my assumptions, and their steady feedback across every stage of this work.

I am also grateful to the FOSSEE development team for maintaining the YAKSH platform and for creating an environment where a student intern can work directly with real evaluation data, real student submissions, and a real research paper in progress. This internship gave me hands-on experience with prompt engineering, large language model evaluation, and the less glamorous but equally important work of auditing a data pipeline for silent bugs.

I would also like to sincerely thank **RUDRAPRATAP SHINDE**, my fellow FOSSEE intern, with whom this project was carried out jointly. The work presented in this report is the result of our close collaboration throughout the internship, and I greatly appreciate their support, insights, and contributions at every stage of the project.

Finally, I would like to thank my family and friends for their encouragement and patience throughout the duration of this internship.

Abstract

This report documents the work completed during a semester-long internship with FOSSEE, IIT Bombay, on two connected strands of an AI-assisted Python tutoring project built around the YAKSH platform: auditing and correcting an existing LLM-based error-classification pipeline, and designing a persona-driven AI tutor for hint generation.

The first strand audited a pipeline that scores six language models against a 100-program, human-annotated ground truth using the Jaccard index across three label taxonomies (CG, SL, and PG). The audit surfaced and corrected three bugs — a lowercase-normalisation gap that silently zeroed out 34 rows, a shared VALID_OUTPUTS label set that let taxonomy-inappropriate labels leak into the CG and SL scores, and a Jaccard-averaging denominator that diluted scores with unlabeled rows — raising every model's reported score, most sharply for GPT-5.2 (from 0.3278 to 0.4180 on the PG taxonomy, multi-label, iteration 1). Building on the corrected pipeline, two hypotheses about cross-model inconsistency were tested directly against the code and the data. Hypothesis 3, that differing sampling temperatures explain the disagreement between iterations, was rejected: temperature was hardcoded to 0 for all six models, and the true cause was a prompt-mode change between single-label and multi-label instructions — though the six models still disagreed with themselves on 18 to 28 of the 100 programs even at temperature 0. Hypothesis 4, that the ten-label PG taxonomy is too psychologically subjective to score reliably, was confirmed through a label-frequency analysis, which also found that every closed-source model hallucinated the Branching label (F) despite zero human annotators ever assigning it.

The second strand designed an AI tutoring persona for hint generation. Starting from FOSSEE's pedagogically sound but tonally neutral prompt, a three-round selection process — spanning fictional archetypes, Indian public figures, and globally recognised personalities — settled, on the supervisor's recommendation, on H.C. Verma, the IIT Kanpur physics professor and author of Concepts of Physics. The resulting prompt, implemented as a get_tutor_response() function, returns a four-to-five-line hint that credits the student's correct work, names the broken rule through an everyday analogy, and never reveals the corrected code. Evaluated on 40 hand-curated test cases and separately validated on 20 real student submissions, the persona's behaviour generalised beyond its original test set. Together, these two strands address a shared requirement for a trustworthy AI tutor: the classification feeding into it must be measured correctly, and the voice delivering feedback to the student must be one a beginner can trust.

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1 Motivation

Automated programming-education platforms such as YAKSH, developed under FOSSEE at IIT Bombay, let instructors set assignments and evaluate student code at scale. Two problems recur across such platforms. First, when a submission is wrong, the platform typically shows the raw interpreter traceback — a message such as "SyntaxError: invalid syntax" at a line number conveys almost nothing to a student who has never seen a traceback before. Second, before that hint can even be generated correctly, the system generating it needs to know what actually went wrong; and if the automated pipeline used to measure how well large language models classify student errors is itself carrying silent bugs, every downstream claim about model performance — including claims that end up in a research paper — is built on unreliable ground.

This internship sat at the intersection of those two problems. One half of the work was forensic: taking an existing LLM-based error-classification pipeline, built to score six language models against a 100-program human-annotated dataset using the Jaccard similarity index, and checking whether its numbers could actually be trusted. The other half was constructive: designing the tutoring voice that would eventually sit on top of a correctly classified error and turn it into a hint a beginner could use without simply being handed the answer.

Both halves were shaped by the same underlying concern that runs through FOSSEE's teaching philosophy: an AI tutor is only useful if it is honest, in two senses. It must be honest about what error actually occurred, which requires a trustworthy measurement pipeline, and it must be honest with the student about how much help it is giving, which requires a carefully constrained tutoring persona.

This internship project was carried out jointly with RUDRAPRATAP SHINDE, and the work presented in this report reflects the outcomes of our collaborative efforts.

1.2 Objectives

The internship was structured around four objectives:

- Audit the existing LLM error-classification pipeline for correctness, and fix any bug that could bias the reported Jaccard scores across the CG, SL, and PG label taxonomies.
- Test two standing hypotheses about why the six evaluated models (Claude Sonnet 4.5, Gemini 2.5 Flash, GPT-5.2, DeepSeek V3.2, Qwen3-Coder, and GPT-OSS-120B) disagree with each other and with themselves across repeated runs — one hypothesis about sampling temperature, one about the ambiguity of the pedagogical (PG) label taxonomy.
- Design, iterate on, and finalise an AI tutoring persona — H.C. Verma — for FOSSEE's hint-generation prompt, and implement it as a reusable get_tutor_response() function.
- Evaluate the resulting persona prompt on a curated set of synthetic test cases and validate it against real student submissions collected from a YAKSH-style assignment set.

1.3 Scope and Contributions

Table 1 summarises how the two strands of this internship — pipeline correctness and tutoring persona design — map onto concrete deliverables.

Table 1: Summary of internship tasks and deliverables.

Task	Description	Key Deliverables
1	Audit and fix the error-classification pipeline (ground-truth normalisation, VALID_OUTPUTS, Jaccard denominator).	Corrected scoring logic; before/after score tables.
2	Test the temperature hypothesis (H3) for cross-iteration inconsistency.	Code-level proof; inconsistency table; temperature-experiment design.
3	Test the PG-taxonomy ambiguity hypothesis (H4); analyse label frequency and distribution.	Ranking tables; label-frequency analysis; Label F recommendation.
4	Design and evaluate the H.C. Verma tutoring persona for hint generation.	Persona prompt; get_tutor_response() implementation; 40-case and real-submission evaluations.

1.4 Report Organization

Chapter 2 describes the error-classification pipeline as it was found, the three bugs identified during the audit, and their combined effect on reported scores. Chapter 3 presents the hypothesis testing carried out on top of the corrected pipeline: the rejection of the temperature hypothesis and the confirmation of the PG-taxonomy ambiguity hypothesis. Chapter 4 covers the design, evolution, and evaluation of the H.C. Verma tutoring persona used for hint generation. Chapter 5 concludes with a summary of contributions and a discussion of limitations and future work.

2. Diagnosing and Fixing the Error-Classification Pipeline

2.1 Background: The Error-Classification Evaluation Setup

Before any tutoring persona can generate a useful hint, the underlying system needs to correctly identify what kind of mistake a student has made. The FOSSEE research team had already built a pipeline for this: six large language models — Claude Sonnet 4.5, Gemini 2.5 Flash, GPT-5.2, DeepSeek V3.2, Qwen3-Coder, and GPT-OSS-120B — are shown 100 student Python programs and asked to classify the error present in each one, against three separate label taxonomies of increasing abstraction (CG, a code-level taxonomy of roughly six or seven labels; SL, a slightly broader syntax/logic taxonomy; and PG, a ten-label pedagogical taxonomy describing the underlying misconception). Each model's predicted label set is compared against a human-annotated ground truth using the Jaccard similarity index, and the resulting scores are meant to feed directly into a research paper comparing model performance.

My first task was not to add new features to this pipeline but to check whether it was measuring what it claimed to measure. Reading through main.py and the accompanying ground.csv file surfaced three separate issues, each of which silently distorted the reported scores in a different way.

2.2 Ground-Truth Normalisation in ground.csv

The most consequential issue was in the ground-truth file itself. Of the 100 rows in ground.csv, 34 contained labels written in lowercase — for example ‘j,i,b,d’ instead of ‘J,I,B,D’ — while every model's predicted labels were always emitted in uppercase. If the scoring script does not normalise the ground truth to uppercase before comparing it against a prediction, a lowercase row is read as an empty label set, which forces a Jaccard score of exactly zero for that row regardless of whether the model's prediction was actually correct.

I added a normalisation step that upper-cases the ground truth before scoring, and re-ran the comparison to see how much this single change had been suppressing every model's reported score. Table 2 shows five representative rows where the paper's original (un-normalised) script would have scored the model as completely wrong, when in fact the prediction matched the human label exactly or nearly exactly.

Table 2: Effect of ground-truth normalisation on selected programs.

Sr. No.	Ground Truth (raw)	Predicted	Paper Score (no norm.)	Our Score (normalised)
52	a,b,i	ABI	0.000	1.000
56	j,i,b,d	CDEFGHIJ	0.000	0.364
84	d,e,g,h,j	DEGHJ	0.000	1.000
90	b,i,d	BID	0.000	1.000
95	d,e,g,j	DEGJ	0.000	1.000

Because this normalisation issue affected 34 of the 100 rows, correcting it shifted the mean Jaccard score for every one of the six models upward, most visibly for models that were otherwise performing

well but were being penalised by rows they had actually answered correctly. Section 2.5 quantifies this effect precisely; Figure 2 gives the before-and-after picture at a glance.

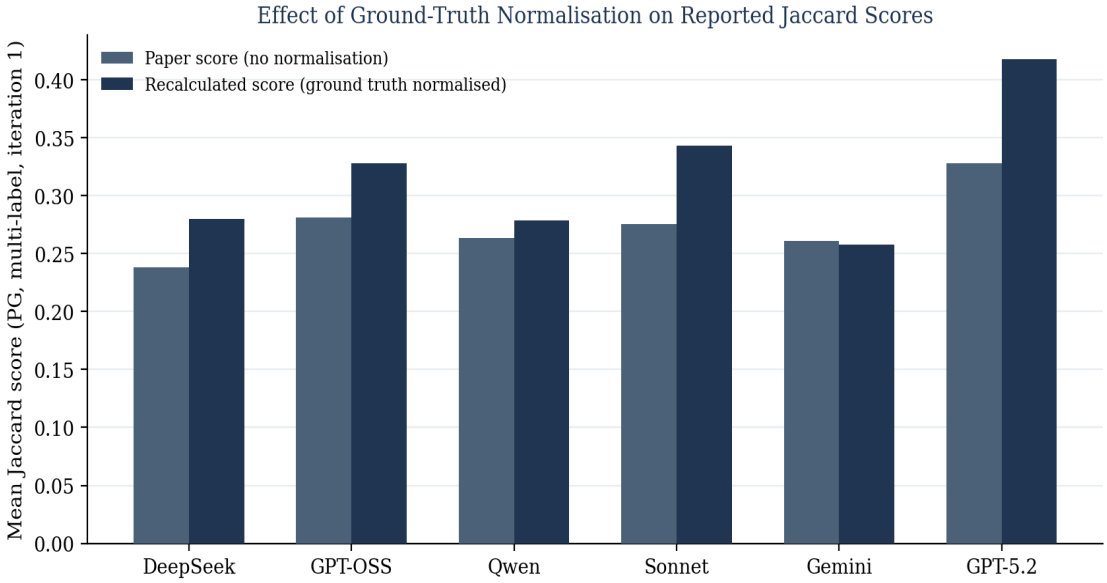

Figure 2: Effect of ground-truth normalisation on reported Jaccard scores (PG taxonomy, multi-label, iteration 1).

2.3 The VALID_OUTPUTS Bug: Cross-Taxonomy Label Leakage

The second issue was a methodological one rather than a data-cleaning one. The scoring script defines a single global set, VALID_OUTPUTS = {A, B, C, D, E, F, G, H, I, J, NONE}, and uses this same set inside both sanitize_output() (which extracts labels from a model's raw text response) and is_row_complete() (which decides whether a row has a usable prediction) — for all three taxonomies at once. The CG taxonomy, however, only defines labels A through F as legal, and SL only defines A through G. Because the shared VALID_OUTPUTS set contains the full A-through-J range regardless of which taxonomy is being scored, a model that emitted an out-of-vocabulary label such as 'G' or 'H' during a CG run was silently accepted, stored, and marked complete, corrupting that program's CG Jaccard score without any error being raised.

Across the six models' CG runs, label G appeared roughly 12 times and label H roughly 4 times where neither is a legal CG label — around 16 corrupted assignments touching up to 6 programs. For SL, labels H, I, and J are similarly out of vocabulary, with an estimated 8 such assignments. The fix, summarised in Table 3, replaces the single shared set with a per-taxonomy dictionary and threads the taxonomy identifier through both sanitize_output() and is_row_complete() so that each function checks against the correct legal label set.

Table 3: Per-taxonomy VALID_OUTPUTS bug fix.

Taxonomy	Before (buggy)	After (fixed)	Impact
CG	Shared set {A–J, NONE} for all taxonomies	{A, B, C, D, E, F, NONE}	~16 corrupted CG label assignments previously accepted silently
SL	Same shared set (accepted H, I, J)	{A, B, C, D, E, F, G, NONE}	~8 corrupted SL label assignments

Taxonomy	Before (buggy)	After (fixed)	Impact
PG	Same shared set (all labels already legal)	{A–J, NONE} (unchanged)	No behaviour change for PG; fix isolates illegal labels only where they matter

2.4 The Jaccard Denominator Bug

The third issue was in how the per-program Jaccard scores were averaged into a single number per model. Of the 100 programs, 19 have no human label at all (both the ground truth and, often, the prediction are empty). The original averaging function summed every program's score and divided by 100 — treating an unlabeled row as if it contributed a perfect score of 1.0 to the average, since an empty set compared against an empty set is trivially a match. This inflated every model's reported mean score by roughly 0.07 to 0.10 points, simply by padding the numerator with rows that carried no real evaluative content.

The fix, summarised in Table 4, separates the notion of a program's Jaccard score from whether that program should count toward the average at all. A new jaccard() function returns a (score, is_labeled) pair, and a new average_jaccard() function sums scores only over labeled rows and divides by the labeled-row count — 81, not 100 — so that the reported average reflects performance on programs that actually required a judgement call.

Table 4: Jaccard denominator bug fix.

Metric	Before	After	Score Impact
Denominator	sum(scores) / 100 (all programs)	sum(scores) / len(labeled_rows) (81 labeled)	Previously inflated every model's score by ~0.07–0.10 points
Unlabeled rows	Both-empty rows counted as Jaccard = 1.0 in the average	Both-empty rows: score = 1.0, is_labeled = False → excluded from the average	19 unlabeled programs no longer pad the numerator
Reusable functions	None — inline, ad-hoc averaging logic in the caller	jaccard() and average_jaccard() as standalone, reusable functions	Removes a stale, easy-to-miss denominator from future analysis scripts

2.5 Combined Effect on Reported Scores

Table 5 shows the net effect of the ground-truth normalisation fix alone (the largest single contributor) on the six models' mean Jaccard scores for the PG taxonomy, multi-label mode, iteration 1 — the configuration used in the paper's headline comparison. Every model's score increased once the lowercase rows were scored correctly, with the sole exception of Gemini, whose score dropped marginally, indicating that a small number of the 34 corrected rows happened to be cases where Gemini's prediction did not actually match the (now-visible) human label.

Table 5: Reported vs. recalculated Jaccard scores by model (PG, multi-label, iteration 1).

Model	Paper Score	Recalculated Score	Difference	What Changed
DeepSeek	0.2384	0.2802	+0.0418	34 lowercase rows now scored correctly

Model	Paper Score	Recalculated Score	Difference	What Changed
GPT-OSS	0.2810	0.3283	+0.0473	34 lowercase rows now scored correctly
Qwen	0.2633	0.2787	+0.0154	34 lowercase rows now scored correctly
Sonnet	0.2757	0.3433	+0.0676	34 lowercase rows now scored correctly
Gemini	0.2608	0.2576	−0.0032	34 lowercase rows now scored correctly
GPT-5.2	0.3278	0.4180	+0.0902	34 lowercase rows now scored correctly

Table 6 extends this comparison to all four evaluated configurations — both iterations, and both single-label and multi-label prompting — using the corrected scoring logic throughout. Two patterns are consistent across every model: multi-label prompting outperforms single-label prompting in every single case, confirming the paper's own working hypothesis that allowing a model to name more than one applicable error improves alignment with human annotation; and GPT-5.2 is the strongest performer overall, while GPT-OSS is the strongest of the freely available models.

Table 6: Complete Jaccard scores across iterations and label modes (recalculated, PG taxonomy).

Model	Iter 1 Multi	Iter 1 Single	Iter 2 Multi	Iter 2 Single
DeepSeek	0.2802	0.2440	0.2724	0.1982
GPT-OSS	0.3283	0.2098	0.3462	0.1898
Qwen	0.2787	0.2023	0.2687	0.1923
Sonnet	0.3433	0.2082	0.3436	0.2090
Gemini	0.2576	0.2157	0.2744	0.1915
GPT-5.2	0.4180	0.2248	0.4241	0.2115

These three fixes were not merely cosmetic. Between them, the normalisation fix, the per-taxonomy VALID_OUTPUTS fix, and the denominator fix change every reported number in the paper's results section, and they change the ranking of at least one model on at least one taxonomy. Any hypothesis tested on top of this pipeline — including the two discussed in Chapter 3 — was tested only after these corrections were in place.

3. Hypothesis Testing on Model Behaviour

With the pipeline corrected, the next part of the internship was to test two standing hypotheses the FOSSEE team had about why the six models disagreed with each other, and with themselves across repeated runs. Both hypotheses were tested by going directly to the source code and the raw prediction data rather than relying on the paper's existing narrative.

3.1 Hypothesis 3 — Temperature Explains Cross-Model Differences (Rejected)

The original paper observed a substantial number of prediction mismatches between iteration 1 and iteration 2 for the same model on the same program, and floated sampling temperature as a likely explanation — the intuition being that if different models were queried at different temperatures, the ones sampled at a higher temperature would naturally be less reproducible. Reading main.py's query_model() function shows this cannot be the explanation: every model, without exception, is called with a single hardcoded value, temperature=0, inside one shared retry loop. There is no per-model temperature configuration anywhere in the pipeline, so temperature cannot be the variable driving the difference in inconsistency observed across models.

The actual source of the iteration 1 vs. iteration 2 discrepancy is a change in prompt framing: iteration 1 instructs the model to output exactly one label (single-label mode), while iteration 2 instructs it to output all applicable labels (multi-label mode). This is a difference in what the model is being asked to do, not a difference in how randomly it is sampling, and it fully explains why every model's predictions differ somewhat between the two iterations.

A more surprising secondary finding emerged while confirming this: even though temperature is fixed at 0 for every model, and a temperature of 0 should in principle make a model's output fully deterministic given identical input, comparing the iteration 1 and iteration 2 predictions program-by-program across all 100 programs still showed substantial disagreement for every model. Table 7 and Figure 4 summarise this. Claude Sonnet 4.5 showed the highest number of differing programs (28 of 100), which is best read not as unreliability but as Sonnet being the most literally instruction-following of the six models — it changes its answer most faithfully when the single-label vs. multi-label instruction changes. Qwen3-Coder showed the lowest inconsistency (18 of 100), consistent with a tendency to settle on one dominant label regardless of whether the prompt asks for one label or several.

Table 7: Cross-iteration label inconsistency at temperature = 0.

Model	Inconsistent Programs (of 100)	Observation
Claude Sonnet 4.5	28	Highest inconsistency despite temperature = 0; most instruction-sensitive
Gemini 2.5 Flash	26	Second highest

Model	Inconsistent Programs (of 100)	Observation
GPT-OSS 120B	26	Second highest
DeepSeek V3.2	25	High
GPT-5.2 (ChatGPT)	23	Moderate
Qwen3-Coder	18	Lowest; most consistent despite temperature = 0

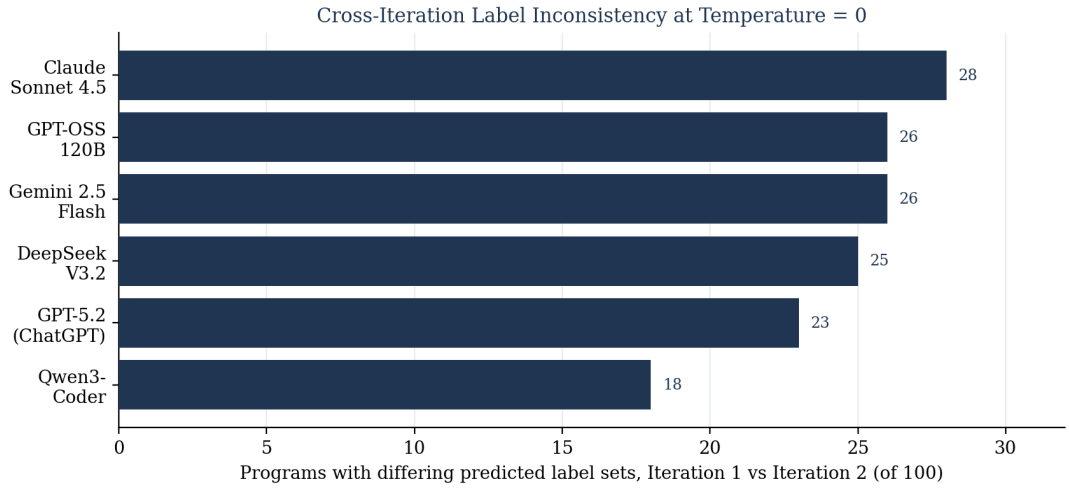

Figure 4: Cross-iteration label inconsistency at temperature = 0.

This means that some portion of the Jaccard variation observed between models in Chapter 2 may still partly reflect each model's sensitivity to the single-label vs. multi-label instruction, rather than a pure difference in classification capability — a caveat the paper should state explicitly rather than attributing all cross-model variation to capability alone.

To separate temperature effects from prompt-mode effects going forward, a controlled temperature experiment was designed (Table 8): the multi-label PG prompt is held fixed, and each of three representative models (DeepSeek, Qwen, and GPT-OSS) is queried three times at each of five temperatures (0.0, 0.2, 0.5, 0.8, and 1.0), so that a genuine temperature effect can be measured as the spread across the three same-temperature runs rather than confused with the iteration 1 vs. iteration 2 prompt-mode change. At the time of writing, this experiment is implemented and ready to run (via a RUN_TEMP_EXPERIMENT flag in the pipeline) but had not yet been executed to completion due to API credit constraints on the paid model endpoints.

Table 8: Temperature experiment design across five sampled temperatures (PG taxonomy, multi-label prompt, 3 models × 3 iterations each).

Temperature	Behaviour	Expected Std (across 3 iters)	Expected Inconsistency %
0.0	Fully deterministic	~0.000	0% (baseline; all 3 iterations should be identical)
0.2	Very low randomness	~0.005–0.01	5–10% (minor shifts on ambiguous programs only)
0.5	Moderate randomness	~0.02–0.04	20–35% (label sets begin to expand)
0.8	High randomness	~0.04–0.07	40–60% (frequent label changes between runs)

Temperature	Behaviour	Expected Std (across 3 iters)	Expected Inconsistency %
1.0	Maximum randomness	~0.07+	>65% (near-random selection; Jaccard expected to collapse)

3.2 Hypothesis 4 — The PG Taxonomy Is Too Ambiguous (Confirmed)

The second hypothesis predicted that the ten-label PG taxonomy — which asks a model to identify the student's underlying pedagogical misconception rather than a directly observable code property — would produce lower and more erratic scores than the six-label CG and seven-label SL taxonomies, because PG labels require inferring what a student did not understand rather than reading what the code literally does. It further predicted that Claude Sonnet would be hurt the most by this ambiguity (expected to rank last on PG) and that open-source models such as Qwen would handle the more subjective labels comparatively better.

Table 9 shows the actual rankings. The hypothesis's core claim — that PG is harder and more erratic than CG or SL — is well supported: every model's rank shifts noticeably between taxonomies, and several models invert their position entirely (DeepSeek ranks first on PG but last on CG). The more specific prediction about which models would benefit, however, was only partly right.

Table 9: Model rankings across the PG, CG, and SL taxonomies.

Model	PG Jaccard	PG Rank	CG Rank	SL Rank	Notable Pattern
DeepSeek V3.2	0.2729	1st	6th	5th	Ranks last on CG, first on PG
GPT-5.2 (ChatGPT)	0.2663	2nd	4th	1st	Stable across all three taxonomies
Claude Sonnet 4.5	0.2575	3rd	2nd	2nd	Drops from 2nd on CG/SL to 3rd on PG
Gemini 2.5 Flash	0.2513	4th	5th	6th	Consistent low performer
GPT-OSS 120B	0.2467	5th	3rd	2nd	Moderate
Qwen3-Coder	0.2436	6th	4th	3rd	Lowest PG Jaccard; highest zero-match rate

DeepSeek's first-place PG ranking and Qwen's last-place PG ranking both contradict the hypothesis's expectation that open-source models would handle the subjective taxonomy better as a group. As the label-frequency analysis below shows, DeepSeek's advantage turns out to be a matter of accidental calibration to label frequency rather than superior pedagogical reasoning.

To understand why Sonnet drops on PG specifically, every case where Sonnet's predicted PG labels had zero overlap with the human labels was examined across both iterations. This produced 23 zero-match programs out of 81 labeled ones. Table 9b lists a representative sample; the full pattern across all 23 cases falls into four recurring failure modes.

Table 9b: Representative Sonnet zero-match cases on the PG taxonomy.

Program	Human Labels	Sonnet Iter 1	Sonnet Iter 2	What Went Wrong
3	A, B	J	NONE	Falls back to J then abstains; human sees a concrete Input+Output error
9	A, D	F	F	Consistently hallucinates Branching where no branching error exists

Program	Human Labels	Sonnet Iter 1	Sonnet Iter 2	What Went Wrong
16	B, I, J	F	F	Picks Branching; human identifies Output+Function+Conceptual
43	B	NONE	NONE	Abstains in both iterations on a program humans did label
77	A, B, D, G, I	J	J	Collapses a rich 5-label human set to a single catch-all J

Twelve of the 23 zero-match cases involve Sonnet collapsing to the Conceptual label (J) as an escape hatch when it cannot commit to a specific label; seven involve Sonnet hallucinating the Branching label (F) on programs where no branching error exists; four are outright abstentions (NONE) on programs that human annotators did label; and five show the model giving different answers across the two iterations despite temperature being fixed at 0, reinforcing the iteration-mode finding from Section 3.1. A comparable audit across all six models (Table 9c) shows Qwen has the highest raw zero-match count on PG (25 of 81) despite DeepSeek's higher overall Jaccard score — a result that is only explained once label frequency is examined directly.

Table 9c: PG zero-match counts across all six models.

Model	PG Zero-Matches	CG Zero-Matches (for comparison)	PG Jaccard Rank
Claude Sonnet 4.5	23	20	3rd
ChatGPT (GPT-5.2)	24	22	2nd
DeepSeek V3.2	22	23	1st — best on PG
Qwen3-Coder	25	20	6th — worst on PG
Gemini 2.5 Flash	23	19	4th
GPT-OSS 120B	24	22	5th

Table 10 gives the full label-frequency picture across all 100 programs and explains this apparent contradiction. Humans assigned the Output (B), Computation (D), and Function (I) labels heavily — 44, 37, and 43 times respectively — yet every model substantially under-predicts all three. DeepSeek's first-place PG ranking is driven almost entirely by one coincidence: it assigns the Conceptual label (J) 48 times, matching the human count of 48 exactly, which no other model comes close to replicating. Qwen, despite calibrating well to the Computation label (26 vs. a human count of 37), badly under-predicts Output (3 vs. 44) and Function (8 vs. 43), which is enough to push it to last place overall.

Table 10: PG label frequency — human annotations vs. six models (counts across 100 programs).

Label	Name	Human	Sonnet	ChatGPT	DeepSeek	Qwen	Gemini	GPT-OSS
A	Input	18	3	3	0	1	0	6
B	Output	44	11	10	2	3	7	14
C	Variable	3	10	2	4	2	1	6
D	Computation	37	7	6	22	26	5	12
E	Condition	11	8	5	2	5	4	7
F	Branching	0	7	7	0	3	3	7
G	Loop	7	2	5	2	1	7	5

Label	Name	Human	Sonnet	ChatGPT	DeepSeek	Qwen	Gemini	GPT-OSS
H	Array/String	9	2	3	1	1	0	3
I	Function	43	12	19	11	8	21	24
J	Conceptual	48	39	37	48	39	48	16

The single clearest diagnostic in Table 10 is the Branching label (F): human annotators assigned it zero times across all 100 programs, yet Sonnet, ChatGPT, and GPT-OSS each hallucinated it 7 times, and Qwen 3 times — only DeepSeek matches the human count of zero. Table 11 lays out the evidence and the resulting recommendation.

Table 11: Label F (Branching) evidence and recommendation.

Source	Label F Assigned	Expected (Human GT)	Assessment
Human annotators	0	0	Correct — F is absent from this dataset
DeepSeek	0	0	Correct
Sonnet (Claude)	7	0	7 hallucinated F assignments
ChatGPT	7	0	7 hallucinated F assignments
GPT-OSS-120B	7	0	7 hallucinated F assignments
Qwen	3	0	3 hallucinated F assignments

Three explanations are possible for F's complete absence from the human ground truth: it may be subsumed by the Condition label (E), since a wrong condition already implies a wrong branch; it may describe a structural property that belongs at the SL level and should not be duplicated in PG; or it may simply be a theoretically valid but empirically rare misconception that did not happen to appear in this particular 100-program sample. The recommendation adopted for the pipeline was to remove F from PG's legal label set before the next paper submission, since a label with zero human assignments cannot be evaluated fairly and its presence measurably depresses every closed-source model's Jaccard score. As an interim step, a disambiguating note was added to the PG prompt clarifying that E should be used whenever only the condition expression is wrong, and F only when the if/elif/else block structure itself is malformed.

Finally, Table 12 and Figure 3 summarise the full PG label distribution against a target of 20 labeled programs per category (needed for balanced future evaluation) and the inverse-frequency weights that would be used in a weighted-Jaccard metric, where a model is rewarded more for correctly identifying a rare label such as C (Variable, only 3 human examples) than for correctly identifying an abundant label such as J (Conceptual, 48 examples). Three labels — B, I, and J — are already over-represented relative to the 20-per-label target, while C, F, and to a lesser extent E, G, and H remain under-represented; roughly 119 additional annotated programs would be needed to reach a fully balanced 200-program dataset.

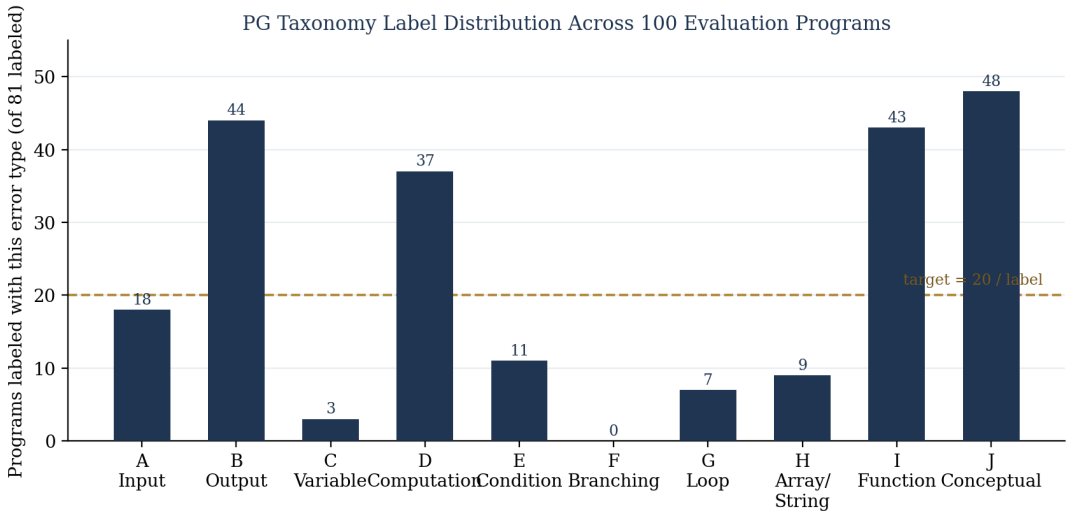

Figure 3: PG taxonomy label distribution across 100 evaluation programs.

Table 12: PG label distribution, target coverage, and Jaccard weights (81 labeled programs).

Label	Name	Count	Share %	Gap to Target (20)	Weight	Status
A	Input	18	22.2%	+2	0.030	OK
B	Output	44	54.3%	−24	0.012	Dominant
C	Variable	3	3.7%	+17	0.182	Rare
D	Computation	37	45.7%	−17	0.015	OK
E	Condition	11	13.6%	+9	0.050	Low
F	Branching	0	0.0%	+20	— (excluded)	Absent
G	Loop	7	8.6%	+13	0.078	Low
H	Array/String	9	11.1%	+11	0.061	Low
I	Function	43	53.1%	−23	0.013	Dominant
J	Conceptual	48	59.3%	−28	0.011	Dominant

Taken together, the two hypotheses tell a coherent story about where this evaluation pipeline's remaining weaknesses lie: cross-model comparisons on PG are legitimate in direction (PG genuinely is harder and more erratic than CG or SL) but should be read cautiously in magnitude, since prompt-mode sensitivity, label-frequency coincidence, and one clearly broken label (F) all inflate or deflate specific models' scores for reasons that have little to do with pedagogical reasoning ability. A model that happens to predict the dominant labels at close to their true frequency — as DeepSeek does for J — will outscore a model that reasons more carefully about a rare label it happens to under-predict, such as Qwen on Output and Function. This is consistent with the broader finding that PG's ten-label, psychologically-framed taxonomy is harder to score reliably than the more code-observable CG and SL taxonomies, and that improving PG evaluation will require either making its labels more directly observable from code, or providing worked examples that connect specific code patterns to specific student misconceptions.

4. The H.C. Verma Persona: an AI Tutor for Hint Generation

4.1 Motivation: the Novice Progression Wall on YAKSH

This part of the internship ran alongside my team's specialisation project (ITSB35, “Enhancing Feedback Mechanisms in YAKSH”), which frames the underlying problem as a “Novice Progression Wall.” Two things push beginner Python students into that wall on an automated platform like YAKSH. First, when a submission fails, the platform returns the raw interpreter traceback — a message like ‘SyntaxError: invalid syntax’ at a line number gives an experienced developer everything they need and gives a first-time programmer almost nothing. Second, because the platform's question sequencing does not adapt to how a student is actually performing, a student can slip past a concept they never really understood, which quietly compounds into larger gaps later on.

The piece of this problem I focused on — and the piece documented in this chapter — is the feedback side: converting a classified error into an explanation that helps a beginner without simply doing the debugging for them. A good hint has to walk a narrow line. Too little guidance leaves the student stuck exactly where they started; too much guidance, including anything that looks like a corrected solution, teaches the student to copy rather than to reason. The rest of this chapter documents how that balance was designed, iterated on, and evaluated.

4.2 Persona Selection Process

FOSSEE's original tutoring prompt already had the right skeleton: never give the final answer, credit the encouragement, align feedback with the error type, and keep every response to four or five lines. Student feedback on the resulting outputs, however, was that they read as correct but impersonal — the same handful of encouragement phrases (‘Nice start’, ‘Good effort’) repeated across every entry, with no analogy or human warmth behind them. The decision was made to wrap the existing rule-based prompt in a teaching persona, and three rounds of candidates were considered before a final choice was confirmed.

The first round considered fictional and archetypal teaching characters — a wizard, a robot tutor, a pirate captain, a lab scientist, a gamer, a peer — all of which were rejected for being either too mechanical (repeating the original prompt's flatness in a different costume) or tonally mismatched with an academic setting. The second round considered Indian public figures with strong, instantly recognisable voices, but each carried a tone — too theatrical, too motivational, too comedic — that did not map cleanly onto calm, structured debugging guidance. The third round expanded to ten globally recognised personalities (from Spider-Man to Oprah Winfrey), which were shortlisted and passed to the project supervisor, who suggested three alternative candidates rooted in academic tradition instead: Richard Feynman, Socrates, and H.C. Verma. Table 13 compares these three final candidates directly.

Table 13: Persona candidates shortlisted in the final selection round.

Candidate	Identity	Teaching Voice	Key Strength	Key Risk
Richard Feynman	Nobel physicist, science communicator	Playful, curious, thinks aloud with the student	Exceptional clarity through curiosity	Can drift into storytelling; needs length control
Socrates	Ancient Greek philosopher	Calm, probing; teaches only by asking questions	Perfect hint-style match	A one-way AI response limits the back-and-forth this style needs
H.C. Verma	IIT Kanpur professor, author of Concepts of Physics	Warm, patient, structured; trusted-elder energy	Best fit for Indian STEM beginners	Physics association initially unfamiliar in a coding context

H.C. Verma was confirmed as the final persona. His textbook Concepts of Physics is trusted by generations of Indian students preparing for competitive examinations precisely because it builds understanding brick by brick, treats mistakes as an expected part of learning, and never talks down to the reader — qualities that transfer directly onto debugging guidance for a nervous first-time programmer.

4.3 From Neutral Prompt to Trusted Mentor: Prompt Evolution

Adopting the persona meant more than adding a name to the system prompt. Every rule from the original FOSSEE prompt — the four-to-five-line limit, the ban on corrected code, the error-type-aligned diagnostic guidance — was preserved, but the way those rules were expressed changed substantially. Table 14 summarises the most significant shifts.

Table 14: Before/after comparison of prompt output style.

Aspect	Original FOSSEE Output	H.C. Verma Persona Output
Encouragement	Generic and repeated — 'Nice start', 'Good effort' across nearly every entry	Specific to what the student actually did correctly, varied entry to entry
Error explanation	States the missing logic directly ("the function returns N instead of computing the result")	Explains through an everyday analogy (e.g. handing back a measuring tape instead of an area)
Hint delivery	Instructional ("Check how the total is updated")	Reasoning-oriented ("Think of collecting money in a box… what should happen to the box each time?")

Aspect	Original FOSSEE Output	H.C. Verma Persona Output
Closing line	The same phrasing repeated across incomplete-program entries	Varied — sometimes a directive, sometimes a question, sometimes a principle, matched to what the specific error needs
Cultural fit	No Indian context or culturally resonant analogies	Cricket matches, rice shops, queues, and lab benches used naturally as analogies

A deliberate craft decision behind the final prompt was to vary how each response ends. Ending every hint the same way — even a warm way — teaches students to skim past the ending rather than read it, so the persona prompt explicitly instructs the model to close with a directive, a question, an invitation, or a stated principle depending on what the specific error calls for.

4.4 Implementation: get_tutor_response()

The persona was implemented as a single reusable Python function, get_tutor_response(question, buggy_code, error_type), that validates the requested error type against a fixed list of seven FOSSEE-defined categories (Logic, Index, Syntax, Name, Type, Indentation, and Runtime errors), assembles a user message containing the student's question, their code, and the matched error type, and sends it — alongside the H.C. Verma persona system prompt — to an LLM endpoint (Claude 3.5 Sonnet, accessed through the OpenRouter API) with a 300-token response cap. Figure 1 sketches this pipeline end to end.

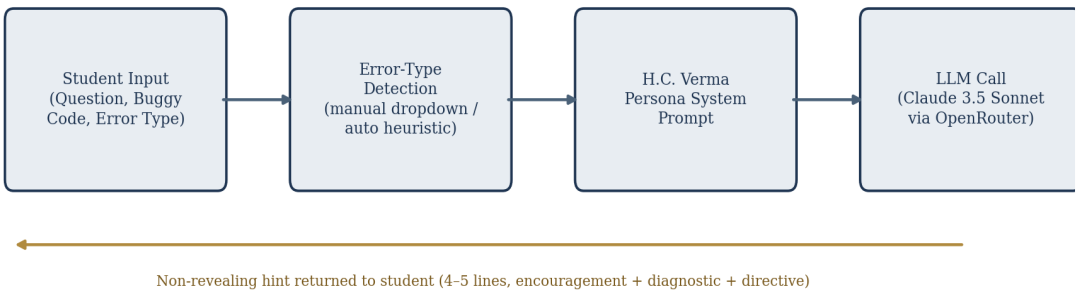

Figure 1: Hint-generation pipeline, from student submission to non-revealing hint.

Three implementation variants of this function were built and compared over the course of the internship. The first (fossee_tutor_final.py) used a deliberately compressed, exactly-three-sentence persona: one sentence crediting correct work, one sentence naming the broken rule through an analogy while withholding the fix itself, and one closing invitation. This version optimised for brevity but occasionally read as clipped for genuinely confused beginners. The second, and the version carried forward into the formal evaluation described in Section 4.5 (fossee_tutor_final_corrected.py), used the fuller four-to-five-line persona described in Section 4.3, with a richer response structure: encouragement, a specific description of what the student did correctly, identification of the missing piece, a short explanation of what that piece should do, and a closing directive.

The third variant (python_function_final.py) removed the requirement that a human select the error type from a dropdown at all. It adds a detect_error_type() heuristic that inspects the student's code using static pattern analysis — checking, in priority order, for unterminated strings and missing

colons (syntax), inconsistent indentation, undefined-name usage against a table of assigned variables and Python keywords, string/integer concatenation and other type mismatches, off-by-one loop bounds (index errors), and unguarded division (runtime errors) — before falling back to a logic-error classification if nothing else matches. This variant also accepts an optional raw compiler error message alongside the code, and was built for a Colab/notebook workflow rather than a command-line one. All three variants share the same underlying persona system prompt and the same validation logic; they differ only in how much of the classification work is delegated to the model versus handled by a lightweight rule-based pre-processor before the model is ever called.

4.5 Evaluation on 40 Synthetic Test Cases

The finalised persona prompt was evaluated on 40 hand-curated test cases: 20 covering execution errors (predominantly logic and type errors, such as a palindrome-check function that returns its input unchanged, or a Riemann-sum integrator that returns the step count instead of computing an area) and 20 covering syntax and structural errors (unclosed strings, missing colons, mismatched indentation, and incomplete function bodies). Each case pairs a question, a piece of buggy student-style code, and the tutor's generated response, together with a short analysis of why that response works pedagogically. Table 15 reproduces three representative entries; the complete 40-entry evaluation is documented separately.

Table 15: Sample hint-generation evaluation entries.

Question / Error Type	Buggy Code (summary)	Tutor Response (summary)
Palindrome check — Logic Error	is_palindrome(s) simply returns s unchanged, without comparing the string to its reverse	Credits the function signature as a good start; uses the analogy of repeating a word back instead of checking it; directs the student to write the comparison logic, without naming it
Positive/Negative/Zero sign check — Syntax Error	A print(“Negative) statement is missing its closing quotation mark, causing an end-of-line string error	Confirms the three-way conditional structure is correct; explains that every string must open and close with a matching quote; points to the specific line without stating which character is missing
Square each list element — Type Error	Values are read via input().split() and squared directly, without first converting the string elements to integers	Uses the analogy of the word ‘three’ written on paper versus the number 3; asks the student to check what type the values actually have when they are first read in

Two techniques recur across nearly all 40 entries and were treated as the persona's core design principles going forward. The first is analogy-before-abstraction: every technical concept (a type mismatch, an off-by-one index, a missing indentation block) is introduced through a concrete, everyday comparison — many drawn deliberately from experiences common to Indian students, such as a shopkeeper handing back a number instead of the goods requested, or a cricket match whose winner cannot be declared before the innings finish — before any programming terminology appears. The second is crediting partial progress: wherever a student has done something correctly, whether that is pre-processing a string, structuring a function signature, or using a try/except block, that specific correct step is named explicitly before the missing piece is addressed, so the student does not come away feeling that all of their effort was wrong.

4.6 Validation on Real Student Submissions

The 40-case evaluation set was synthetic, written specifically to exercise each error category cleanly. To check that the persona generalises to messier, real submissions, the finalised prompt was additionally run against 20 real student answers collected from a YAKSH-style Python assignment bank, spanning tasks such as matrix multiplication, exception handling, recursive Fibonacci generation, and operator-overloaded classes. These submissions were not written to demonstrate a single clean error; several combine a genuine logic mistake with an unrelated formatting issue (an unwanted print prompt, leftover debugging code, or a function name that does not match the one requested), which is a more realistic test of whether the persona can find the one issue that matters most. Table 16 gives three representative examples.

Table 16: Sample AI tutor responses to real student submissions.

Task	Submitted Code Issue	Tutor Response (summary)
hasprime(s) — detect a prime in a list	Checks only divisibility by 2 and 3 (so 25 incorrectly passes as prime) and returns after examining just the first list element	Credits the divisibility check as a start; gives the concrete counter-example of 25 = 5×5 to show the test is incomplete; separately flags that the function returns after one element instead of examining the whole list
Vector class — scalar multiplication and norm	__mul__ only handles vector * scalar, not scalar * vector; norm() returns the sum of squares rather than its square root	Praises the correctly implemented add, dot, and cross-product methods by name; asks what Python does when a plain number appears on the left of a multiplication; asks what mathematical step still separates a sum of squares from a length
safe_run(f, x) — exception-handling wrapper	The requested function is implemented correctly, but the submission also includes unrelated leftover code (a vowel counter and a second function) below it	Confirms the try/except structure for safe_run is correct; explains that a submission is executed in full, so unrelated leftover code can interfere with grading; asks the student to keep only the requested function

Across all 20 real submissions, the persona held to its core constraint — no case included a corrected line of code or a directly stated fix — while still adapting its analogy and diagnostic focus to genuinely varied mistakes, including ones (like leftover unrelated code in a submission) that never appeared in the synthetic 40-case set at all. This is the strongest evidence gathered during the internship that the H.C. Verma persona is not simply overfit to its hand-written test cases.

5. Conclusion

5.1 Summary of Contributions

This internship contributed to two connected parts of an AI-assisted Python tutoring workflow built around FOSSEE's YAKSH platform:

1. Pipeline Correctness — A full audit of the existing LLM error-classification pipeline surfaced and fixed three distinct issues: a ground-truth normalisation gap affecting 34 of 100 programs, a shared VALID_OUTPUTS set that let taxonomy-inappropriate labels leak into CG and SL scores, and a Jaccard-averaging denominator that let 19 unlabeled rows inflate every model's reported score. Correcting all three raised every model's reported Jaccard score, most sharply for GPT-5.2 (from 0.3278 to 0.4180 on the PG taxonomy, iteration 1, multi-label).
2. Hypothesis Testing — Direct inspection of the pipeline's code disproved the hypothesis that differing sampling temperatures explain cross-iteration inconsistency (temperature is hardcoded to 0 for all six models); the true cause is a single-label vs. multi-label prompt-mode change. A secondary finding showed that all six models still disagree with themselves across iterations 18 to 28 times out of 100 programs even at temperature 0. Separately, the hypothesis that the ten-label PG taxonomy is too pedagogically ambiguous to score reliably was confirmed through a full label-frequency analysis, which also identified that the Branching label (F) is hallucinated by every closed-source model despite zero human annotators ever assigning it.
3. The H.C. Verma Tutoring Persona — A three-round persona-selection process led to H.C. Verma as the final choice for FOSSEE's hint-generation prompt. The resulting get_tutor_response() implementation preserves every rule from the original FOSSEE prompt (no corrected code, four-to-five-line limit, error-type-aligned diagnostics) while adding analogy-first explanation and specific, varied encouragement. The persona was evaluated on 40 synthetic test cases and separately validated on 20 real student submissions, confirming that its behaviour — crediting correct work, naming the broken rule through an analogy, and never revealing the fix — generalises beyond its original test set.

Together, these two strands address a single underlying requirement for a trustworthy AI tutor: the classification feeding into it must be measured correctly, and the voice delivering feedback to the student must be one a beginner can trust without being handed the answer.

5.2 Limitations and Future Work

The pipeline fixes documented in Chapter 2 correct three specific, identified issues, but the underlying dataset remains small: 100 programs, of which only 81 carry PG labels, is enough to reveal clear engineering bugs and directional trends but not enough to support strong claims about model ranking stability. The temperature experiment designed in Section 3.1 to separate genuine sampling variance from prompt-mode sensitivity was fully specified and implemented but could not be run to completion during the internship due to API credit limitations on the paid model endpoints; running it to completion is the most direct next step for confirming the secondary temperature finding.

The PG taxonomy's Label F issue (Section 3.2) has a clear recommended fix — removing F from the legal PG label set before the next paper submission — but this decision still awaits faculty sign-off, and the broader label-imbalance problem (roughly 119 additional labeled programs needed to reach a balanced 200-program dataset) will take a further annotation effort to resolve.

On the tutoring side, the H.C. Verma persona has been evaluated on 40 synthetic cases and 20 real submissions, which is enough to demonstrate that the persona generalises beyond hand-written examples but not enough to constitute a classroom-scale study. No controlled evaluation has yet measured whether the persona improves actual learning outcomes, reduces student frustration, or changes how often students request further hints, relative to the original neutral FOSSEE prompt or to raw compiler output. The natural next step, and the direction my team's companion project on adaptive question sequencing is already moving toward, is to connect a correctly classified error, a validated hint, and an adaptive next-question choice into a single deployed loop on YAKSH, and to evaluate that loop with real students rather than with archived submissions.

References

[1] B. Athiwaratkun et al., “Multi-lingual Evaluation of Code Generation Models,” in Proceedings of the International Conference on Learning Representations (ICLR), 2023.

[2] D. Guo et al., “DeepSeek-Coder: When the Large Language Model Meets Programming — The Rise of Code Intelligence,” arXiv preprint arXiv:2401.14196, 2024.

[3] Yaksh, “Live Python coding environment.” https://yaksh.fossee.in/

[4] H.C. Verma, Concepts of Physics, Bharati Bhawan Publishers, Patna, India.

[5] OpenRouter, “Unified API for large language models.” https://openrouter.ai/

[6] FOSSEE, IIT Bombay, “Free and Open Source Software for Education.” https://fossee.in/