Diagnosing CS1 Misconceptions with LLMs: A Pilot Study on Alignment, Variability, and Prompt Sensitivity

1. Introduction

As LLM-based tutoring tools move from experimental use to routine adoption in CS1 courses, understanding their instructional reliability has become an urgent concern. Large language models (LLM) are rapidly entering computing education as debugging assistants, code explainers, and novice-support tools, yet concerns persist in their instructional reliability and reasoning quality [3], [4], [5], [6], [7]. Prior studies show that while LLMs can often generate syntactically correct code, they may misidentify misconceptions or provide explanations that are fluent but conceptually misaligned with student reasoning [3], [4], [6], [10], [11]. For instructors, the central pedagogical question is not whether an LLM can produce a correct solution, but whether it can correctly diagnose the conceptual error a novice has made an ability foundational to effective formative assessment [1], [2].

Misconceptions in CS1, such as off-by-one errors, accumulator misuse, and misunderstandings of immutability, are well-documented and structurally consistent across student populations [8], [9]. These errors often reflect deeper conceptual gaps rather than surface-level syntax mistakes, and accurate diagnosis is critical for remediation [1], [2], [8], [9]. However, LLM-generated feedback varies substantially depending on prompt design, model family, and inference conditions, raising concerns about diagnostic stability in educational settings [4], [6], [11].

LLMs may appear competent, but their disagreement on core misconceptions undermines their usefulness in formative CS feedback without instructor oversight. This work contributes to an empirical baseline by examining whether widely used LLMs converge or diverge when diagnosing the same novice programming misconceptions. Rather than benchmarking model performance, we focus on diagnostic alignment, cross-model variability, and prompt sensitivity. Our goal is depth rather than breadth: a detailed analysis of individual model diagnoses that expose where and why diagnostic disagreement emerges, informing the design of future misconception-aware tutoring systems. To situate this investigation, we next review prior work on LLMs in computing education and established research on CS1 misconceptions, highlighting the absence of empirical studies examining cross-model diagnostic agreement on identical student errors [3], [5], [6].

2. Related Work

LLMs have been explored in computing education for code generation, debugging assistance, hint generation, and automated feedback [3], [4], [5], [6], [7], [10]. Although these systems often perform well on code correctness tasks, multiple studies report instability in reasoning and explanation quality, particularly when models are asked to justify or diagnose errors [4], [6], [11]. Research on AI-supported feedback consistently emphasizes the need for semantic evaluation and instructor oversight [1], [2], [10].

Decades of CS education research have documented recurring misconceptions in introductory programming across loops, indices, accumulators, and mutability [8], [9]. Traditional automated misconception detection systems rely on rule-based approaches or supervised machine learning, requiring stable features and reliable annotations [8], [9]. Recent work highlights that LLM reasoning may vary even when outputs appear superficially similar [11], but few studies examine misconception-level disagreement across multiple models on identical student inputs. This pilot directly addresses this gap by examining how multiple widely used LLMs diagnose the same instructor-verified CS1 misconceptions under controlled prompting conditions.

3. Methods

3.1 Study Design and Rationale

We conducted a qualitative-quantitative pilot using 10 novice-authored Python snippets representing canonical CS1 misconception categories. The dataset was intentionally small to enable fine-grained, per-response qualitative analysis and to isolate diagnostic variability across models without confounding task heterogeneity.

We examined three mainstream LLMs as students typically encounter them:

ChatGPT (two prompt formulations: Prompt 1 and Prompt 2)

Claude (Prompt 1)

GitHub Copilot (Prompt 1)

Claude and Copilot were evaluated only under Prompt 1; prompt-sensitivity analysis therefore applies only to ChatGPT.

3.2 Materials: Student Snippets and Instructor Anchor

The dataset consists of 10 short Python functions implementing a reverse_string(s) routine. Each snippet contains a single misconception validated by an instructor with over 10 years of CS1 teaching experience.

Instructor-labeled misconception categories included:

Off-by-one loop errors, Immutable string misunderstandings, Accumulator re-initialization, Incorrect slicing semantics, Wrong return type or variable, Incomplete iteration, Pointer mis-initialization

The instructor annotated each snippet with a bug location, concise diagnosis, and pedagogical hint. These labels serve as pedagogical anchors rather than an absolute oracle.

Severity assignment: Each snippet was labeled as minor (1) or moderate (2). A two-level severity scale was used to support consistent coding and interpretation across reviewers.

3.3 Prompts and Inference Parameters

Two prompts were used verbatim to ensure reproducibility.

Prompt 1 (ChatGPT, Claude, Copilot):

(Full prompt text provided in Appendix B.)

Prompt 2 (ChatGPT only):

(Full prompt text provided in Appendix B.)

All models were queried using single-shot deterministic responses under provider default inference settings. Exact temperature and seed metadata were not fully logged for all providers; this limitation is addressed in Section 6. All models were accessed via their free web interfaces: ChatGPT (free tier) and Claude (free tier).

3.4 Coding, Semantic Matching, and Reliability

Two independent coders (the author and an external reviewer) with CS1 teaching experience mapped each LLM diagnosis onto the instructor taxonomy. Coding proceeded independently, followed by discussion-based resolution. When necessary, the instructor's original explanation was consulted as a tiebreaker.

Pre-discussion agreement was 80%. Given the exploratory nature of this pilot and the focus on qualitative diagnostic patterns rather than statistical inference, we report percent agreement and use consensus mappings for all subsequent analyses.

Semantic normalization treated synonymous phrasing (e.g., "off-by-one" vs. "skips final index") as conceptual matches.

3.5 Outcome Measures

We report:

Alignment: Semantic match between LLM diagnosis and instructor label

Pairwise agreement: LLM-to-LLM consistency

Variability:

Variability = (number of unique mapped diagnoses across models) − 1

Prompt sensitivity: ChatGPT Prompt 1 vs. Prompt 2

Severity distribution

4. Results

Using the procedures described above, we evaluated whether large language models converge on instructor-verified misconception diagnoses and where their reasoning diverges. Across the ten

novice-authored code snippets, we examined alignment with instructor labels, pairwise agreement between models, diagnostic variability per snippet, and sensitivity to prompt formulation. Even within this tightly controlled CS1 task, the results reveal meaningful differences in how models diagnose underlying conceptual errors. These findings quantify where LLM diagnoses align with pedagogical expectations and where disagreement emerges, motivating caution in their use for formative feedback.

4.1 Human Coding Reliability

Coders achieved 80% agreement prior to discussion, yielding stable consensus mappings for all analyses.

4.2 Alignment with Instructor Labels

Under Prompt 1, two of the evaluated models exhibited higher diagnostic convergence with instructor-labeled misconceptions, while one model showed greater variability.

ChatGPT (Prompt 1): 50%

Claude (Prompt 1): 90%

Copilot (Prompt 1): 90%

Model alignment percentages were computed by comparing each model's diagnosis independently against the instructor label; Table 1 presents per-submission diagnoses, while alignment counts are aggregated at the model level.

Sub.	Instructor Misconception	Bug Location	ChatGPT Diagnosis	Claude Diagnosis	Copilot Diagnosis	Semantic Match (per-model, aggregated)
1	Off-by-one loop error	Line 3	Loop index off	Loop index off	Loop index off	Yes
2	Immutable string misunderstanding	Lines 3-6	Confusion about immutability	Confusion about immutability	Confusion about immutability	Yes
3	Accumulator re-initialization	Lines 2-4	Wrong output type	Re-initialization inside loop	Re-initialization inside loop	Partial / No
4	Incomplete iteration	Lines 4-5	Loop index off	Generic logic error	Loop index off	No

Sub.	Instructor Misconception	Bug Location	ChatGPT Diagnosis	Claude Diagnosis	Copilot Diagnosis	Semantic Match (per-model, aggregated)
5	Incorrect concatenation order	Lines 3-4	Incorrect concatenation order	Wrong return variable	Incorrect concatenation order	Partial
6	Off-by-one loop error	Line 3	Loop index off	Loop index off	Loop index off	Yes
7	Wrong return type (list vs. string)	Line 4	Wrong return variable	Wrong return variable	Wrong return variable	Yes
8	Wrong return variable	Line 5	Wrong return variable	Incorrect concatenation order	Returns original value	Partial / No
9	Incorrect use of indices / truthiness	Line 4	Incorrect use of truthiness	Loop index off	Re-initialization inside loop	No
10	Loop index off	Lines 3-4	Loop index off	Loop index off	Loop index off	Yes

Table 1. Per-submission instructor misconception labels and LLM diagnoses under Prompt 1. Semantic alignment indicates whether each model's diagnosis matches the instructor's misconception category after coder normalization rules (synonym handling and conservative mapping). Partial indicates the model referenced the correct underlying mechanism but was mapped to a different category under the conservative coding rule. Full student code and model excerpts are provided in Appendix A.

4.3 Pairwise LLM Agreement

Claude-Copilot: 100%

ChatGPT-Claude: 50%

ChatGPT-Copilot: 50%

Although Claude and Copilot produced highly similar diagnostic labels in this pilot, we do not assume shared architectures or training; instead, this convergence indicates consistent labeling for a subset of canonical CS1 misconceptions in this dataset.

4.4 Variability Distribution

Four submissions produced complete diagnostic agreement across all three models, while the majority exhibited at least one dissenting diagnosis. Notably, one submission produced three distinct misconception categories, indicating maximal diagnostic divergence.

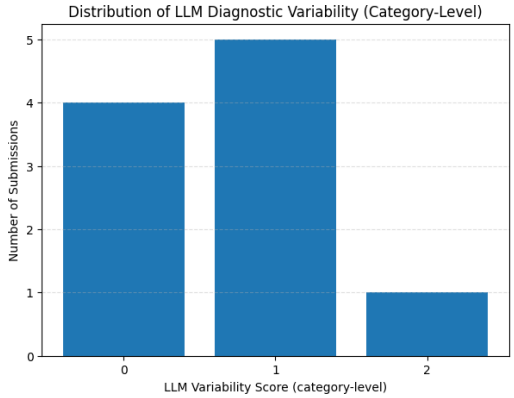


Figure 1. Distribution of diagnostic variability across the 10 submissions under Prompt 1, computed at the *misconception-category level*. Variability is defined as *(number of distinct mapped misconception categories across ChatGPT, Claude, and Copilot − 1)*, where 0 indicates full agreement, 1 indicates one dissenting diagnosis, and 2 indicates three-way disagreement

4.5 Severity Distribution

Instructor-provided severity labels showed:

Minor: 40%

Moderate: 60%

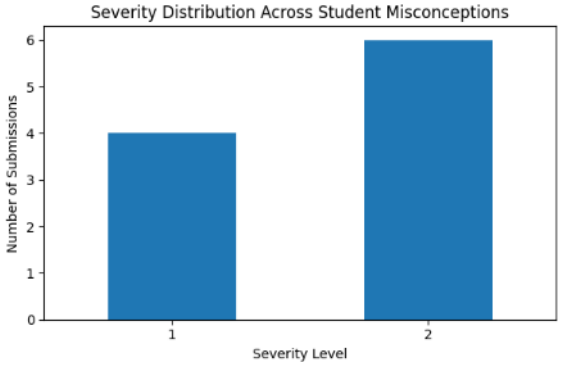


Figure 2. *Instructor-assigned severity distribution across the 10 misconception instances (1 = minor, 2 = moderate), used to contextualize diagnostic disagreement by instructional risk.*

4.6 ChatGPT Prompt Sensitivity

ChatGPT changed its mapped diagnosis on 6 of 10 snippets between Prompt 1 and Prompt 2. Alignment dropped from 50% to 30%, indicating strong prompt dependence.

4.7 Representative Excerpts

Submission 1: All models identified the missing boundary condition, supporting low variability.

Submission 3: Different phrasing masked a shared conceptual issue, requiring semantic normalization.

Submission 4: Models emphasized different mechanisms, reflecting genuine diagnostic divergence.

Submission 8: Two models failed to identify an incorrect return variable, masking the core misconception.

5. Discussion

Taken together, the results support this paper's central claim: although LLMs often appear competent when explaining or debugging novice code, their disagreement on core CS1 misconceptions limits their reliability as formative feedback tools without instructor oversight. Even within a constrained task and a small set of canonical errors, models diverged in how they diagnosed underlying conceptual flaws, particularly for misconceptions requiring multi-step reasoning. This variability is pedagogically consequential, as effective formative feedback depends not only on plausibility but on accurately targeting the learner's misconception [1], [2].

The findings further show that diagnostic convergence is misconception dependent. While some error types elicited stable agreement across models, others produced divergent diagnostic framings that could mislead learners or obscure the underlying conceptual issue [3], [4], [6]. In instructional contexts, such inconsistency risks reinforcing incorrect mental models or eroding student trust in feedback systems, undermining their pedagogical purpose [1], [2], [8], [9].

Consistent with the study's pilot scope, these results should be interpreted as descriptive patterns of diagnostic behavior rather than comparative performance claims.

6. Limitations and Future Work

As an early-stage pilot, this work is intentionally scoped to surface diagnostic patterns rather than to support statistical generalization. The dataset size reflects a deliberate design choice to enable close examination of model reasoning across individual misconceptions; as a result, the findings should be interpreted as exploratory baselines rather than performance rankings.

Inference metadata such as temperature and random seeds were not fully logged for all model providers, which limits claims about stochastic stability. However, the core contribution of this study lies in identifying cross-model diagnostic disagreement, which emerged consistently across multiple misconception categories and is unlikely to be attributable solely to sampling noise.

Prompt sensitivity was examined only for ChatGPT, as Claude and Copilot were accessed under a single shared prompt in this pilot. Accordingly, prompt-level conclusions are restricted to ChatGPT and should not be generalized across models.

Future work will expand the dataset to a larger and more diverse collection of CS1 submissions, incorporate full inference logging, and conduct multi-sample stability analyses. We also plan to explore ensemble and self-consistency methods, as well as prompts that enforce structured, pedagogy-aligned reasoning to improve diagnostic reliability.

7. Conclusion

This work provides an empirical baseline for evaluating LLM diagnostic reliability in CS1 contexts. Although some models show strong convergence with instructor reasoning, others exhibit substantial prompt sensitivity and conceptual drift. Across models, disagreement on deeper misconceptions highlights the risk of deploying LLMs as autonomous formative feedback tools. LLMs can support learning, but without structured prompts and instructor oversight, they remain unreliable for diagnosticians. This pilot lays out the groundwork for transparent, auditable AI-supported tutoring systems in computing education.

References

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

[2] V. J. Shute, "Focus on formative feedback," *Review of Educational Research*, vol. 78, no. 1, pp. 153-189, 2008.

[3] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman, "Studying the effect of AI code generators on supporting novice learners in introductory programming," in Proc. CHI Conf. Hum. Factors Comput. Syst., Art. no. 455, 2023.

[4] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in Extended Abstracts of the CHI Conf. Hum. Factors Comput. Syst., Art. no. 332, 2022.

[5] S. Lau and P. J. Guo, "From 'Ban It Till We Understand It' to 'Resistance Is Futile': How university programming instructors plan to adapt as more students use AI code generation and explanation tools," in Proc. ACM Conf. Int. Comput. Educ. Res. (ICER), vol. 1, pp. 106-121, 2023.

[6] A. Hellas, J. Leinonen, S. Sarsa, C. Koutcheme, L. Kujanpää, and J. Sorva, "Exploring the responses of large language models to beginner programmers' help requests," in Proc. ACM Conf. Int. Comput. Educ. Res. (ICER), vol. 1, 2023.

[7] C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, and I. Gazit, "Taking flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools," ACM Queue, vol. 20, no. 6, pp. 35-57, Jan. 2023.

[8] A. Luxton-Reilly, "Learning to Program Is Easy," *Proc. ITiCSE*, 2016.

[9] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas, "A multi-national study of reading and tracing skills in novice programmers," *ACM SIGCSE Bulletin*, vol. 36, no. 4, pp. 119-150, 2004.

[10] T. Phung, V.-A. Pădurean, A. Singh, C. Brooks, J. Cambronero, S. Gulwani, A. Singla, and G. Soares, "Automating human tutor-style programming feedback: Leveraging GPT-4 tutor models for hint generation and GPT-3.5 student models for hint validation," *arXiv preprint*, 2024.

[11] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *arXiv preprint* arXiv:2205.11916 [cs.CL], 2022. *(Accepted at NeurIPS 2022).*

[12] I. Azaiz, N. Kiesler, and S. Strickroth, "Feedback-generation for programming exercises with GPT-4," *arXiv preprint* arXiv:2403.04449 [cs.AI], 2024. *(Accepted at ITiCSE 2024).*

Appendix A

Submission	**Student Code**	**Misconception (taxonomy)**	**Bug Location**	**Hint**	**Reasoning**	**ChatGPT Feedback**	**Claude Feedback**	**Copilot Feedback**
1	def reverse_string(s): result = ""\nfor i in range(len(s)-1): result = s[i] +	Loop index off / Off-by-one	Line 3	Check loop covers all	Last character skipped;	Loop index off; last char skipped	Loop index off (3); last char skipped	Loop index off; final char skipped

Submission	Student Code	Misconception (taxonomy)	Bug Location	Hint	Reasoning	ChatGPT Feedback	Claude Feedback	Copilot Feedback
	result\nreturn result			characters	rest correct			
2	def reverse_string(s): for i in range(len(s)): s[i] = s[len(s)-i-1]\nreturn s	Immutable string misunderstanding	Lines 3-6	Python strings are immutable	Strings can't be modified in place.	Confusion about immutability	Confusion about immutability (5); immutable	Confusion about immutability; strings can't be assigned
3	def reverse_string(s): result = None\nfor ch in s: result = ch + result\nreturn result	Wrong output type / Re-initialization inside loop	Line 2-4	Is initial value compatible with string concat?	Initializing as None breaks concatenation	Wrong return type; None + string invalid	Re-initialization inside loop (4); overwrites result	Re-initialization inside loop; only last char remains
4	def reverse_string(s): result = ""\ni = 0\nwhile i < len(s): result = s[i] + result\ni += 2\nreturn result	Loop index off / Incomplete logic	Line 4-5	Are all characters visited?	Skips every second character	Loop index off; skips chars	Loop index off / slicing syntax; skips chars	Loop index off; every 2nd char skipped
5	def reverse_string(s): result = ""\nfor i in range(len(s)): result = result +	Incorrect concatenation order / Broken logic	Line 3-4	Are characters added to front/back?	Adds to end, not front → output unchanged	Incorrect concatenation order	Wrong return variable / returns list?	Incorrect concatenation; builds in same order

Submission	Student Code	Misconception (taxonomy)	Bug Location	Hint	Reasoning	ChatGPT Feedback	Claude Feedback	Copilot Feedback
	s[i]\nreturn result							
6	def reverse_string(s): result = ""\nfor i in range(len(s)-1, 0, -1): result += s[i]\nreturn result	Loop index off	Line 3	Include first char when iterating backward	Stops before 0; first char missing	Loop index off; first char missing	Loop index off (3); first char missed	Loop index off; first char not included
7	def reverse_string(s): output = []\nfor ch in s: output.append(ch)\nreturn output	Wrong return variable / Wrong type	Line 4	Should function return string or list?	Collects characters in list, not string	Wrong return variable	Wrong return variable / no accumulation	Wrong return variable; function returns list
8	def reverse_string(s): result = ""\nfor i in s: result = i + result\nreturn s	Wrong return variable / Broken logic	Line 5	Check variable being returned	Reversal done in result, but returns original	Wrong return variable; result ignored	Incorrect concatenation order; returns s instead	Returns s instead of result
9	def reverse_string(s): result = ""\nfor i in range(len(s)):\nif s: result = s[i] + result\nreturn result	Incorrect use of truthiness / Re-init in loop	Line 4	Does condition change character accumulation?	Always true; loop logic unaffected, shows confusion	Incorrect use of truthiness	Loop index off / re-init; condition redundant	Re-init inside loop; condition meaningless

Submission	Student Code	Misconception (taxonomy)	Bug Location	Hint	Reasoning	ChatGPT Feedback	Claude Feedback	Copilot Feedback
10	def reverse_string(s): result = ""\nfor i in range(0, len(s), -1): result += s[i]\nreturn result	Loop index off / Incorrect accumulation	Line 3-4	Will the loop execute with these start/step ?	Loop never runs; direction misaligned	Loop index off; loop never runs	Loop index off (3); negative step prevents execution	Loop index off; invalid range, loop fails

Appendix B

Prompt 1:

You are a CS assistant tutor helping a student debug Python code. Do not provide a full solution. Instead, give hints and explanations that guide the student toward correcting the bug.

Course context: Introductory programming (loops, strings, accumulators).

Student code:
{student_code}

Misconception taxonomy (for labeling):

1. Wrong return variable
2. Incorrect concatenation order
3. Loop index off-by-one
4. Re-initialization inside loop
5. Confusion about immutability

Instructor rubric:
- Identify the bug location
- Explain why it is a bug
- Provide a hint without giving the solution
- Connect the explanation to a course concept
- Maintain a supportive tone

Task:

1. Identify the bug in the code.
2. Describe the misconception type.
3. Provide a hint that will guide the student.
4. Briefly explain why the bug causes incorrect behavior (2-3 sentences).

Output format:
Bug Location: <line>
Misconception: <taxonomy item>
Hint: <hint text>
Reasoning: <brief conceptual explanation>

Prompt 2:

You are an AI tutoring assistant for an introductory programming course.

You never provide full code solutions.

Your task is to:

1. Identify the student's likely misconception.
2. Provide one hint that moves the student toward the fix.
3. Provide one explanation describing what part of the student's reasoning may be incorrect.
4. Student code:
   {student_code}

Use the following misconception categories as guidance (not an exhaustive list):
- Off-by-one errors
- Incorrect slicing semantics
- Immutable string misunderstanding
- Wrong output type
- Broken algorithm logic
- Incomplete logic
- Incorrect accumulation or loop direction
- Incorrect use of indices or truthiness

Feedback should be under 150 words, supportive in tone, and avoid giving code.