Calibrating LLM Tutor Initiation to Learner Expertise: Proactive and Reactive Support for Computational Thinking

Working paper. Under review at Information Systems Research.
This version: May 19, 2026

Arun Rai
Georgia State University

Liwei Chen
University of Cincinnati

Balasubramaniam Ramesh
Georgia State University

Cynthia Breazeal
MIT Media Lab, Massachusetts Institute of Technology

Yuan Long
Georgia State University

Benjamin M. Ampel
Georgia State University

Andrea Aria
Georgia State University

Abstract

Generative AI (GenAI) has facilitated advanced educational applications, including adaptive tutoring systems. Nevertheless, a central design question in IS research remains unresolved: how should diagnostic responsibility be allocated between the learner and the AI tutor to foster, rather than replace, learner reasoning? This study identifies tutor initiation as a critical design dimension in LLM-based tutoring that redistributes diagnostic responsibility between the learner and the system. Drawing on cognitive load theory's expertise reversal logic, we theorize how tutor initiation interacts with learner expertise and the problem-solving phase to influence computational thinking. This dimension is formalized in the CodeTutor artifact, which serves as a design class for expertise-contingent support systems. A controlled experiment with 90 students revealed no average treatment effect but demonstrated substantial heterogeneity in conceptual reasoning: proactive initiation benefits novices by accelerating error diagnosis, whereas reactive initiation supports advanced learners by preserving self-diagnosis. These effects do not extend to code implementation, where external feedback substitutes for the tutor's diagnostic role. Process-level analysis of interaction traces identifies three mechanisms (help articulation, error recovery, and solution navigation) through which tutor initiation redistributes diagnostic responsibility. The findings indicate that the effectiveness of GenAI tutoring depends on how diagnostic responsibility is allocated between the learner and the system. By conceptualizing tutor initiation as a mechanism for governing cognitive labor during interaction, this study advances research on AI-supported learning and human–AI collaboration.

Keywords: Computational thinking, LLM-based tutoring, AI-supported learning, tutor initiation, learner expertise, human–AI interaction, process analysis

AI Disclosure

Generative AI tools were used for language editing. The authors reviewed all content and take full responsibility for it. The use of large language models as part of the research design (the CodeTutor artifact and the automated scoring pipeline) is described in the paper and Appendix C.

Corresponding author: Arun Rai, Georgia State University, arunrai@gsu.edu

1. INTRODUCTION

Computational thinking (CT) is a foundational capability for programming education and for success in an AI-enabled world (Brennan and Resnick 2012). CT encompasses core processes such as decomposition, abstraction, and algorithm construction, which require learners to actively diagnose problems, identify solution paths, and iteratively refine their reasoning (Tang et al. 2020). Generative AI (GenAI) systems can support these processes by assisting learners during problem-solving. However, such assistance may diminish learners' opportunities to construct solutions independently. Prior research on cognitive offloading indicates that individuals delegate cognitive processes to external resources to reduce effort, particularly during demanding phases such as problem diagnosis and planning (Fan et al. 2025). In GenAI-supported environments, these external resources manifest as AI-generated solutions, and such delegation can result in increased reliance and reduced cognitive engagement, thereby bypassing the diagnostic and reasoning processes essential for developing CT proficiency (Kosmyna et al. 2025). Consequently, a central challenge in enhancing CT among learners is not merely providing AI support but structuring it to preserve and strengthen the learner's cognitive engagement (Roschelle et al. 2025).

Compounding this challenge, opportunities to develop CT are distributed unevenly. Students from underrepresented and socioeconomically disadvantaged backgrounds frequently encounter persistent barriers in introductory programming courses, including limited prior exposure to computing and reduced access to personalized support (Breazeal et al. 2024, Johnson et al. 2007, Walton and Cohen 2007). These disparities are especially pronounced at minority-serving institutions (MSIs), where high drop, fail, withdraw (DFW) rates in gateway courses can limit entry into computing and AI-related careers (Van Dusen et al. 2020). As a result, learners vary substantially in their expertise, particularly in their ability to recognize, diagnose, and respond to difficulties during problem-solving. Addressing these disparities requires scalable support that helps students persist through learning demands while maintaining the cognitive processes essential for CT development.

In practice, such support is often delivered through educational technologies (EdTech), including GenAI-supported learning environments. However, EdTech systems are not often designed to promote the

higher-order reasoning required for CT. Instead, EdTech often emphasizes rapid feedback and error resolution, helping learners fix issues without engaging in deeper problem-solving processes (Gupta and Bostrom 2013, Wambsganss et al. 2025). At the same time, learners require support to make progress on complex tasks, especially when they cannot independently diagnose and resolve difficulties. Such forms of support design create a tension: while assistance can help learners complete tasks, it may also reduce their engagement in the reasoning processes necessary for CT development (Urban et al. 2024; Li et al. 2026).

This tension reframes the design problem from determining what assistance to provide to considering how assistance is delivered during interaction. An important but underexplored dimension is *the initiation of support during problem-solving.* Previous research has emphasized the content or style of support, such as direct explanations or Socratic guidance (Graesser et al. 2012, VanLehn 2011) but has given less attention to the initiation of support. This study conceptualizes this dimension as tutor initiation: whether support is initiated by the tutor (proactive tutor initiation) or by the learner (reactive tutor initiation). Advances in large language model (LLM)-based systems now enable the implementation and study of these distinct modes of tutor initiation in real time.

Tutor initiation determines how responsibility for diagnosing problems is distributed between the learner and the system. Under proactive tutor initiation, the system intervenes when it detects signals of difficulty, potentially reducing the learner's need to identify and articulate problems. Under reactive tutor initiation, learners must recognize and express their own difficulties before receiving support, thereby retaining responsibility for initiating problem diagnosis. This distinction shapes who performs the diagnostic and reasoning processes central to problem-solving. Because CT development depends on learners' engagement in these processes, shifts in diagnostic responsibility directly affect whether learners actively construct understanding or rely on system-provided solutions.

The effects of tutor initiation depend on learner expertise, which shapes the ability to perform diagnostic and reasoning processes independently. Novice learners often lack the metacognitive skills required to recognize and articulate problems, making it difficult for them to initiate help. For these learners, proactive tutor initiation can reduce the diagnostic burden by providing timely guidance when learners are

unable to identify or express difficulties, thereby supporting continued progress (Graesser et al. 2012, Kalyuga et al. 2003). In contrast, more experienced learners are better able to diagnose problems and regulate their own problem-solving processes. For these learners, reactive tutor initiation preserves diagnostic responsibility, allowing them to engage more fully in the reasoning processes that support CT development (Santhanam et al. 2008, Kalyuga et al. 2003). Thus, tutor initiation may differentially support or hinder CT development depending on learners' ability to perform diagnostic processes independently. Building on this logic, our central research question is: *How does the interaction between tutor initiation (proactive versus reactive) and learner expertise shape CT development across phases of problem-solving in introductory programming at MSIs?*

To address this question, a large language model (LLM)-enabled EdTech tutor, CodeTutor, was designed and evaluated to govern tutor initiation while maintaining consistent pedagogical style and curricular content. A multi-method design science approach was employed, integrating discovery, evaluation, and process explanation. During the discovery phase, a series of focus groups, with over 200 students, at a minority-serving institution (MSI) were conducted to identify unmet learning needs in existing GenAI systems, informing the design of CodeTutor. CodeTutor implements two modes of tutor initiation: proactive, in which the system intervenes based on signals of difficulty, and reactive, where learners request assistance. Subsequently, a controlled experiment with 90 students in an introductory programming course examined how tutor initiation interacts with learner expertise to influence CT proficiency. To elucidate these effects, a process analysis of tutoring interactions was conducted, focusing on how learners articulate help requests, recover from errors, and navigate solution paths.

The results indicate that the effect of tutor initiation is contingent on learner expertise. Proactive initiation enhances CT proficiency for learners with lower initial proficiency or limited programming experience, while reactive initiation benefits more advanced learners. Additionally, proactive initiation strengthens learners' ability to integrate core CT components during conceptual reasoning but not during code implementation. Process analysis reveals that these effects stem from differences in diagnostic responsibility. Examination of how learners formulate help requests, recover from errors, and navigate AI-

suggested solutions demonstrates that proactive initiation provides earlier diagnostic framing and supports recovery from error states among novice learners. Conversely, reactive initiation preserves self-diagnosis, productive persistence, and ownership over the solution path for experienced learners.

This study makes three primary contributions. First, it identifies *tutor initiation* as a central design dimension for GenAI-based learning systems, demonstrating how it redistributes diagnostic responsibility between the learner and the system. Second, it shows that the effectiveness of GenAI support is both expertise- and phase-contingent: different initiation modes benefit learners at different levels of initial CT proficiency, with effects concentrated on conceptual reasoning rather than on code implementation. Third, it extends cognitive load theory's expertise reversal logic to GenAI-supported learning by identifying tutor initiation as the design mechanism through which expertise- and phase-contingent effects are enacted, thereby shifting attention from system features to interaction design.

2. THEORETICAL FOUNDATION

2.1 Computational Thinking (CT) as the Learning Target

CT is a foundational problem-solving capability that involves recognizing patterns, abstracting essential structure, decomposing problems, and designing algorithms that integrate these elements into executable solutions (Brennan and Resnick 2012). Programming requires learners to coordinate these processes in real time while interpreting task requirements, debugging code, and revising their reasoning. Prior research on EdTech-mediated learning emphasizes that effective learning depends not only on acquiring individual skills but also on integrating them through active engagement and practice (Gupta and Bostrom 2013, Piccoli et al. 2001). This emphasis on integration aligns with broader findings that complex problem-solving requires the coordination of cognitive and metacognitive processes rather than the isolated execution of skills (Urban et al. 2024).

Developing CT proficiency, therefore, is challenging because learners must simultaneously diagnose problems, evaluate alternative solution paths, and iteratively refine their reasoning during problem-solving. As a result, effective learning depends not only on completing tasks but also on sustained engagement in the diagnostic and reasoning processes that underlie CT.

2.2 AI-Enabled Learning Support and Its Tradeoffs

Given the integrative demands of CT, learners require support to make progress during problem-solving. EdTech systems, increasingly powered by GenAI, provide such support by offering feedback, guidance, and solution suggestions during task performance (Li et al. 2026, Piccoli et al. 2001). When appropriately designed, these systems can enhance engagement and learning by guiding learners through difficult stages of problem-solving (Kim et al. 2025).

However, such support can also introduce unintended consequences. When assistance is provided without appropriate constraints, learners may rely on it in ways that encourage cognitive offloading, delegating key reasoning processes to external systems to reduce effort (Fan et al. 2025). Evidence from GenAI-supported learning environments shows that such reliance can improve task performance while weakening learning outcomes (Li et al. 2026, Lo et al. 2024). This pattern reflects a broader concern that AI-supported problem solving may shift learners' attention toward producing correct outputs rather than engaging in the diagnostic and reasoning processes required for CT. These findings highlight the tension that AI-enabled support can facilitate and substitute for learner reasoning. As a result, the effectiveness of AI tutoring depends on the availability of support and how it is delivered within the learning process.

2.3 Interaction Design: Tutor Initiation as the Distribution of Diagnostic Responsibility

Prior research on EdTech tutoring has primarily focused on what support is provided, such as explanations and hints that guide learners through problem-solving (Graesser et al. 2012, VanLehn 2011). These approaches emphasize the content and style of support, demonstrating how different forms of guidance influence learning outcomes. However, less attention has been paid to how support is delivered within interaction, particularly when it enters the problem-solving process. This interaction-level dimension shapes whether learners engage in diagnosing problems, formulating solution strategies, and regulating their own reasoning. As a result, understanding learning in AI-supported environments requires examining not only what support is provided but also how it is delivered during problem-solving.

Literature in intelligent tutoring systems (ITS) has considered this interaction by examining solicited versus unsolicited help and learners' help-seeking behavior (Aleven et al. 2016). We extend this

inquiry to LLM-based tutors and conceptualize tutor initiation as a key interaction-level intervention: whether the tutor initiates support or the learner does. Under reactive tutor initiation, the system waits for the learner to request assistance, requiring the learner to recognize and articulate difficulties before receiving support. This preserves learner-driven diagnosis and aligns with prior work on self-regulated learning and productive struggle (Aleven et al. 2006, Kapur 2008, Santhanam et al. 2008) but assumes that learners can accurately detect and express their need for help (Nelson-LeGall 1981). Under proactive tutor initiation, the system intervenes based on signals of difficulty, such as repeated errors or extended inactivity. This shifts part of the diagnostic burden from the learner to the system and aligns with prior work on adaptive scaffolding in EdTech (Dever et al. 2024, Winkler et al. 2021).

Tutor initiation distributes diagnostic responsibility between learner- and system-driven processes, affecting engagement during problem-solving. The "assistance dilemma" in intelligent tutoring highlights that more help speeds progress but limits self-diagnosis, while less help preserves agency but risks prolonged struggle (Koedinger and Aleven 2007). In GenAI-supported environments, system-generated solutions can intervene directly, exacerbating this dilemma. Early intervention can hinder reflection and mental model development (VanLehn 2011, Abdel-Karim et al. 2023), while delayed help may leave learners under-guided. Thus, tutor initiation is crucial for balancing system support and learner reasoning.

2.4 An Expertise-Contingent Expectation

The effectiveness of tutor initiation varies with learner expertise. The expertise reversal effect suggests that instructional scaffolds that are beneficial for novices can become redundant or disruptive for more advanced learners (Kalyuga et al. 2003). Recent work conceptualizes expertise reversal as part of a broader element interactivity framework in which the same scaffold may aid or impede learning depending on whether learners can integrate the relevant information and solution steps (Chen et al. 2017). Applied to GenAI tutoring, this perspective implies that proactive and reactive initiation may have different effects depending on learners' ability to perform diagnostic and reasoning processes.

Learners with lower initial CT proficiency or less prior programming experience face concurrent demands of task understanding, constructing code, interpreting errors, and deciding when and how to seek

help. For these learners, proactive initiation serves a compensatory role by providing guidance when they are unable to diagnose problems or formulate effective help requests. This compensatory role of proactive initiation aligns with evidence that structured support and reduced cognitive load disproportionately benefit less experienced learners (Hwang et al. 2021, Li et al. 2024, Muralidharan et al. 2019).

Learners with high initial CT proficiency or programming experience are better able to diagnose problems and regulate their problem-solving processes. For these learners, reactive initiation preserves opportunities for self-diagnosis and sustained engagement in reasoning, allowing them to maintain control over the solution process. Proactive initiation, however, may intervene prematurely, reducing learning opportunities to reflect and refine their mental models (VanLehn 2011).

These differences reflect broader variation in how learners assess their progress and decide when to rely on AI support (Von Zahn et al. 2025). Accordingly, the effectiveness of tutor initiation depends on how well the interaction design aligns with learners' diagnostic capabilities. Consistent with this perspective, research on human-centered AI shows that system effectiveness depends on aligning system behavior with users' cognitive characteristics, with benefits varying according to user experience and trust (Krakowski et al. 2023, Wang et al. 2024). In this expertise-contingent view, tutor initiation serves as the mechanism for allocating diagnostic responsibility, determining whether the learner or the system assumes primary responsibility for diagnosing and responding to problem-solving impasses.

2.5 From Contingent Outcomes to Process Explanation

Although prior arguments establish that tutor initiation shapes learning outcomes, understanding how these effects unfold requires examining learners' interaction processes during problem-solving. If tutor initiation changes the distribution of diagnostic responsibility, its effects should manifest in how learners recognize difficulties, respond to feedback, and incorporate guidance into their evolving solutions (Gupta and Bostrom 2013). In AI-supported learning environments, guidance is dynamically generated and integrated into ongoing task performance, influencing how learners interpret feedback, respond to errors, and refine their reasoning over time (Wambsganss et al. 2025). These interaction processes are not uniform. Learners vary in their ability to recognize when they need help, articulate their difficulties, and effectively

incorporate external guidance into their problem-solving (Dever et al. 2024, Urban et al. 2024). As a result, the effects of tutor initiation are enacted through interaction processes that shape how learners engage with support during problem-solving.

2.6 A Phase-Contingent Expectation

The expertise-contingent effects of tutor initiation are likely to vary across phases of problem solving. CT proficiency develops through phases (conceptualization, planning, implementation, and verification) that impose different cognitive demands and offer different sources of feedback (Polya 1957). The diagnostic burden is most acute during conceptualization and planning, when learners must construct an initial problem representation, identify subgoals, and decide on an approach without external execution feedback. These early phases lack the error signals that later phases provide; learners must detect difficulties through self-monitoring rather than by observing program behavior.

In contrast, the implementation phase introduces a continuous stream of external feedback through compiler errors, runtime exceptions, and test failures. This feedback substitutes for part of the diagnostic work that the tutor initiation would otherwise perform: the integrated development environment (IDE), which provides a centralized interface to write, test, and debug code, itself signals that something is wrong without the learner needing to identify the difficulty independently or the tutor needing to intervene. Where external signals already do diagnostic work, the marginal value of tutor initiation is attenuated. This logic extends prior work on the assistance dilemma (Koedinger and Aleven 2007), in which the same support mode produces different effects depending on which guidance sources are already operating in the learner's environment. This logic also generalizes across tasks. Tutor initiation should matter most when execution feedback is sparse and least when it is dense. The contrast between the conceptual reasoning task (T1) and the code implementation task (T2) reflects this distinction.

We expect tutor initiation effects on CT proficiency to differ across problem-solving phases. Specifically, we expect the moderation between tutor initiation and learner expertise on CT proficiency growth to be more pronounced in the conceptual reasoning phase than in the code implementation phase. We also expect tutor-initiation-driven gains in structural CT integration to concentrate in conceptual

reasoning rather than implementation because the coupling among CT dimensions is most malleable when learners are constructing the conceptual representation rather than executing it.

3. THE DISCOVERY PHASE: FROM LEARNER NEEDS TO DESIGN REQUIREMENTS

3.1 Discovery Process

To translate the theoretical argument into a GenAI-supported learning artifact, we conducted a discovery phase to identify mismatches between the needs of learners and the delivery of support that existing GenAI tutoring systems provide. We drew on focus groups with students enrolled in a gateway Python course at a large urban MSI, and instructor interviews. Participants varied in terms of prior programming experience, allowing us to observe differences in how learners approached problem-solving and engaged with GenAI support. In structured problem-solving sessions, students interacted with existing GenAI systems and an early CodeTutor prototype, enabling direct comparison of support interactions across systems. These sessions generated interaction logs, which, together with instructor input, provided additional insight into how learners recognized difficulties, sought help, and recovered from impasses during problem-solving. Detailed procedural information is reported in Appendix A.

Across these observations, a consistent pattern emerged: learners frequently experienced difficulty during moments of impasse because they were unable to recognize or articulate the source of their difficulty. Students often struggled to formulate effective help requests when they could not diagnose why their solution approach was failing. At the same time, students emphasized the importance of maintaining control over the problem-solving process and preferred support that helped them continue reasoning through problems rather than immediately supplying complete solutions. These observations reveal a mismatch between learner needs and the interaction design of AI tutoring systems. Existing systems rely heavily on learner-initiated help, which assumes the learners can accurately diagnose and articulate their difficulties. This mismatch directly motivated the design requirements for CodeTutor.

3.2 Design Requirements

The discovery phase suggested that effective GenAI tutoring systems must balance timely assistance with learner-driven problem-solving.. To address this challenge, three design requirements for CodeTutor were

identified: (1) support proactive help during impasses, (2) support reactive help during self-regulated problem-solving, and (3) emphasize structural scaffolding over quick fixes. These choices influence how support is timed and structured, impacting learning effectiveness (Shute 2008) and how learners engage with assistance, articulate needs, and respond.

4. THE CODETUTOR ARTIFACT: OPERATIONALIZING TUTOR INITIATION

Building on the identified design requirements, we developed CodeTutor, a GenAI-integrated (GPT-4o) EdTech system that represents a design class for expertise-contingent support. CodeTutor integrates a code editor and chat interface within a unified workspace, along with a control layer that governs when the tutor intervenes, how it scaffolds support, and how responses are anchored in course materials. Figure 1 shows the learner-facing environment used in the study. The interface remained constant across conditions, with only tutor initiation varying (full system architecture in Appendix A Figure A1).

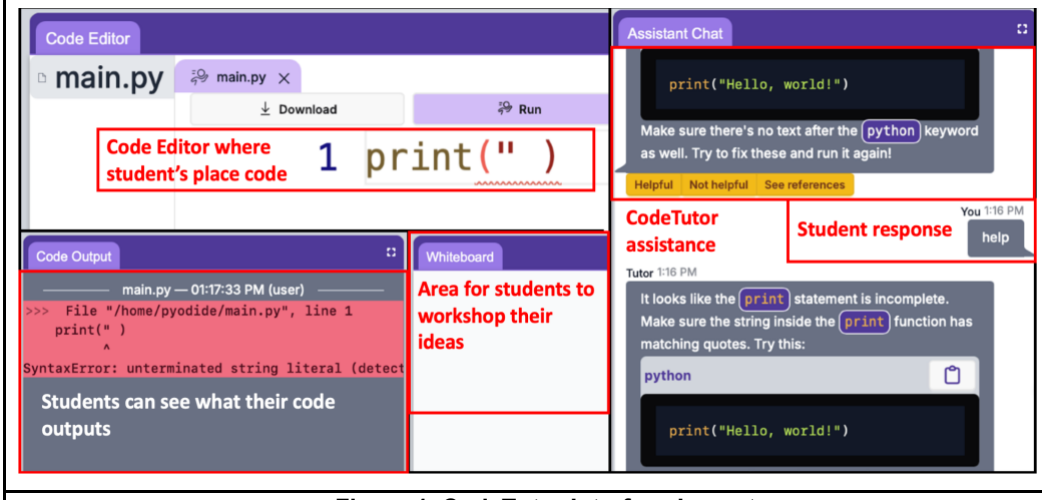


Figure 1. CodeTutor Interface Layout

The design novelty of CodeTutor lies in its initiation logic, which operationalizes expertise-contingent support as a set of actionable design principles. CodeTutor includes two initiation modes. In reactive initiation, the tutor is available when the learner explicitly requests help, preserving the learner's role in recognizing and articulating difficulties. In proactive initiation, the system monitors real-time interaction signals, such as repeated errors, execution failures, or extended inactivity, and initiates support

when those signals indicate sustained difficulty. This design shifts part of the diagnostic responsibility from the learner to the system, aligning support with learners' ability to identify and respond to breakdowns.

Across both modes, CodeTutor employs a consistent scaffolding strategy based on Socratic prompting, providing guidance that emphasizes problem structure rather than direct solutions. The system is constrained from rewriting code or providing complete answers, instead directing attention to decomposition, abstraction, control flow, and data structures. This design ensures that support reinforces reasoning processes central to CT development while holding instructional style constant across conditions. Figure 2 provides illustrative examples of reactive and proactive CodeTutor interactions.

Reactive CodeTutor (initiated by student)	**Proactive CodeTutor** (initiated by tutor)
You: Help me find my errors	**CodeTutor**: Great job using `isdigit()` to identify ratings! To store the ratings, use `rating.append(int(game[i]))` to convert them to integers and add them to your list. Keep up the great work!
CodeTutor: Your `games_list` string is missing a closing quote for `"chess"`. Also, `min(average)` should be `min(ratings_list)`, as `average` is a single number. Fix these errors, and it should work!	**CodeTutor**: Looks like there's a mix-up in calculating the sum! Make sure you're summing the list `rating` as a whole, not individual elements, by using `sum(rating)`. Keep your focus sharp!
You: I'm still doing something wrong	**You**: What should I change
CodeTutor: Currently, your code's input is not being used to calculate the average and minimum ratings. You'll need to parse the `rating` input similarly to `ratings_list` and `games_list`. Update your `ratings_list` and `games_list` by iterating over the `rating` input list, separating the game names from the ratings, and converting the ratings to integers.	**CodeTutor**: Make sure to convert each item to an integer before appending to the `rating` list. Inside your loop, use `rating.append(int(game[i]))` to ensure the sum calculation will be correct.

Figure 2. Illustrative Reactive and Proactive CodeTutor Interactions

To reduce the risk of inappropriate responses, CodeTutor incorporates a retrieval-augmented generation (RAG) pipeline (Gao et al. 2023) anchored in instructor-provided materials. This design keeps GenAI support aligned with course expectations and limits CodeTutor's ability to introduce off-syllabus solutions that may be technically correct but instructionally misaligned. By constraining content and instructional style, the system isolates tutor initiation as the focal design dimension of the study.

5. EXPERIMENTAL EVALUATION

5.1 Evaluation Setting, Participants, and Procedure

We evaluated CodeTutor in an introductory Python programming course at a large public MSI. The course is a required gateway for the Information Systems major and enrolls approximately 220 students per

semester. This setting provides substantial variation in learners' prior programming experience, making it well-suited for examining whether the effects of tutor initiation depend on learner expertise.

Following an IRB-approved protocol, we recruited 94 students. Participation was incentivized through course extra credit, with an alternative assignment available to ensure voluntary participation. Participants were assigned to conditions using stratified randomization on initial CT proficiency and the background characteristics reported in Table 1. After excluding four students who did not complete the second session, the final sample comprised 90[1] students (45 reactive; 45 proactive). Balance checks confirmed that both groups were statistically equivalent before the intervention (Table 1).

Table 1. Sample Characteristics by Treatment Condition

Variable	Category	Full Sample (N = 90)	Reactive (n = 45)	Proactive (n = 45)	*p*
Gender	Female	46 (51.1)	24 (53.3)	22 (48.9)	.68
	Male	44 (48.9)	21 (46.7)	23 (51.1)	
Ethnicity	Non-Hispanic	76 (84.4)	41 (91.1)	35 (77.8)	.18
	Hispanic	12 (13.3)	4 (8.9)	8 (17.8)	
	Not reported	2 (2.2)	0 (.0)	2 (4.4)	
Race	Black or African American	46 (51.1)	26 (57.8)	20 (44.4)	.41
	Asian	24 (26.7)	13 (28.9)	11 (24.4)	
	White	13 (14.4)	5 (11.1)	8 (17.8)	
	More than one race	3 (3.3)	0 (.0)	3 (6.7)	
	Not reported	4 (4.4)	1 (2.2)	3 (6.7)	
Pell eligibility	Pell eligible	44 (48.9)	22 (48.9)	22 (48.9)	1.00
	Not Pell eligible	46 (51.1)	23 (51.1)	23 (51.1)	
Course section	1	15 (16.7)	8 (17.8)	7 (15.6)	.36
	2	17 (18.9)	6 (13.3)	11 (24.4)	
	3	22 (24.4)	10 (22.2)	12 (26.7)	
	4	36 (40.0)	21 (46.7)	15 (33.3)	

The study was conducted over two sessions. In the first session, participants completed a background survey and the pre-test measuring initial CT proficiency. In the second session, conducted two days later, participants engaged in the CodeTutor interaction phase under their assigned initiation mode and then completed the post-test. Each of the pre-test, CodeTutor interaction, and post-test phases consisted of two tasks: T1 (conceptual reasoning), in which participants described the solution logic in natural-language pseudocode, and T2 (code implementation), in which they implemented the solution in Python. Because the pre- and post-tests were completed without AI assistance, this design allows us to assess whether the tutoring interaction led to changes in CT proficiency that persisted beyond the interaction itself.

[1] With N = 90, the study is adequately sized for estimating primary effects but is limited for higher-order interactions and subgroup-specific models. We report standardized effect sizes and bootstrap robustness checks for focal interaction, survival, and mixed-effects estimates, and interpret subgroup patterns with appropriate caution.

Table 2. Study Tasks Across Three Phases			
Phase	**Core Task Logic**	**T1: Natural-Language Reasoning (10 min)**	**T2: Python Implementation (15 min)**
Pre-test	Identify the maximum value	Describe the solution steps in natural language	Write the Python program independently
CodeTutor interaction	Aggregation and comparison task	Describe the solution steps with the assigned CodeTutor mode	Implement the solution with the assigned CodeTutor mode
Post-test	Identify maximum value	Describe the solution steps in natural language	Write the Python program independently
Note: CodeTutor was available only during the middle interaction phase. The pre-test and post-test assess CT proficiency without AI assistance. The three phases used parallel programming tasks that differed in context while preserving underlying reasoning demands across T1 and T2.			

5.2 Experimental Study Design

5.2.1 Treatment: Tutor Initiation Mode

The experiment manipulated only the tutor's initiation mode. In the reactive condition, CodeTutor remained available but intervened only when the learner explicitly requested help. In the proactive condition, CodeTutor monitored interaction signals (e.g., repeated errors, prolonged inactivity) and initiated support when these signals indicated sustained difficulty. We held all other aspects of CodeTutor constant across conditions, including the system prompt[2], instructional style, and course-grounded RAG pipeline. This design isolates tutor initiation as the sole source of variation, allowing differences in outcomes to be attributed to when and how support is initiated.

5.2.2 Student Programming Tasks

To assess changes in CT proficiency, participants completed three parallel programming tasks: (1) a pre-test, (2) a CodeTutor interaction task, and (3) a post-test. CodeTutor was available only during the interaction phase, while the pre- and post-tests were completed independently. This structure allowed us to assess whether learning extends beyond the interaction itself. Each task included two components. T1 required participants to type logic in natural language (i.e., pseudocoding) without feedback, capturing conceptual reasoning. In contrast, T2 required implementation in Python in an IDE with continuous

[2] The CodeTutor system prompt tells the AI to act as a charming, encouraging, hint-providing assistant for an introductory Python course on topics such as user input, loops, conditionals, and basic statistical calculations. The assistant's core role is to offer concise, helpful, and Socratic hints (a maximum of two sentences) to guide students toward completing their programming assignment. CodeTutor is strictly constrained, only providing hints or clarifying a hint up to two times, and is forbidden from responding directly to any other student questions.

compiler and runtime feedback. The tasks differed in surface context but shared comparable underlying logic, enabling consistent assessment of CT proficiency across phases (Table 2 summarizes the tasks).

5.3 Measures

We measure students' CT proficiency using a structured rubric that captures both component skills and their integration during problem-solving (see Table B1 in Appendix B). The rubric operationalizes four core CT dimensions (i.e., pattern recognition, abstraction, decomposition, and algorithmic design) and enables consistent assessment across both natural-language reasoning (T1) and code implementation (T2). This approach follows established principles for operationalizing constructs and measuring them in IS research (MacKenzie et al. 2011). To apply this rubric consistently across submissions, we developed a structured scoring pipeline (Figure 3) that evaluates student responses against dimension-specific criteria.

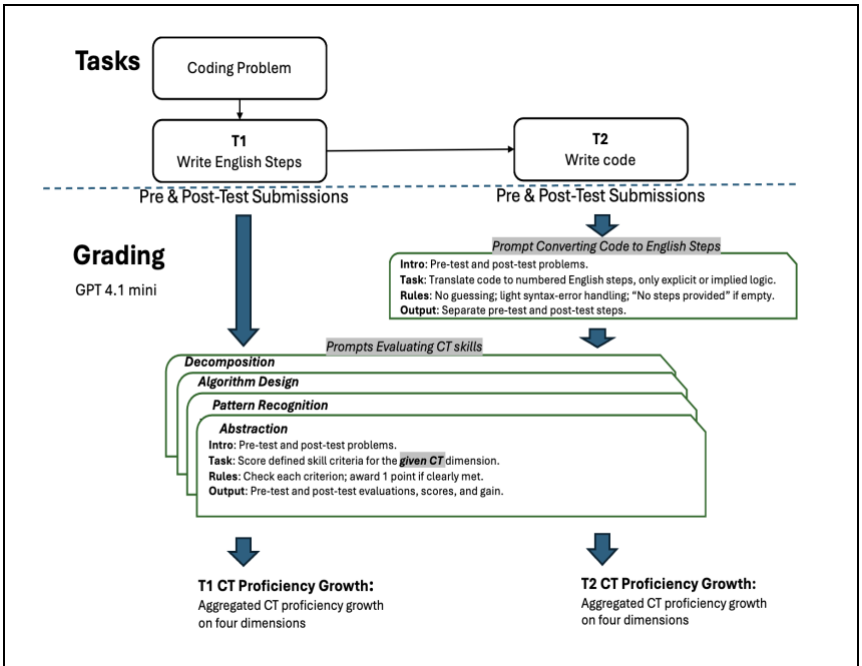


Figure 3. CT Proficiency Scoring Pipeline

Code submissions were translated into comparable logical steps, allowing both T1 and T2 outputs to be assessed using the same rubric. Scores were assigned based on observable evidence for each CT dimension and normalized to ensure comparability across tasks and formats, so that scoring reflects explicit evidence rather than vague or underspecified reasoning (Madaan et al. 2023). Detailed rubric definitions and scoring procedures are provided in Appendices B and C.

We verified this approach through a multi-stage process involving iterative calibration and comparison with independent human ratings (see Appendix C). Both human raters and LLM-based evaluators were blinded to treatment conditions, and the rating rubric was finalized before post-test data collection to mitigate scoring bias. To avoid model contamination, we used a separate LLM for evaluation rather than the one generating student responses. Validation against human ratings shows strong agreement; for example, the decomposition dimension achieved a Cohen's kappa of $\kappa = 0.85$. In addition, comparisons between model-generated and human ratings indicate strong classification performance, with F1 scores ranging from 0.80 to 0.96 across rubric components. These results support the use of the grading pipeline for evaluation, consistent with prior evidence on LLM-based annotation under well-specified prompts (Gilardi et al. 2023; Bhat & Varma 2023).

The pipeline generates three key metrics used in analysis. *Initial CT Proficiency* is based on pre-test performance. *CT Proficiency Growth* captures performance changes from pre-test to post-test, both overall and within each CT dimension. *Structural CT Integration* is assessed through changes in the covariance among CT dimensions over time, reflecting the extent to which learners coordinate multiple CT skills following tutoring intervention. Together, these measures allow us to examine both whether learners improve and how their problem-solving structure evolves following the tutoring intervention. We also included a set of background and contextual variables capturing prior programming experience, self-efficacy, and support-seeking behavior. Measurement items and descriptive statistics for these survey-based constructs are reported in Appendix D.

5.4 Results

Table 3 provides the full pairwise correlation matrices and statistical tests supporting the Structural CT Integration findings.

Table 3. Descriptive Statistics and Correlation Matrix													
Variable	**Mean**	**SD**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1. CT Proficiency Growth (T1)	.00	1.00	1.00										
2 . CT Proficiency Growth (T2)	.00	1.00	.13	1.00									
3.Tutor Initiation Mode	.50	.50	-.03	-.03	1.00								
4.Initial CT Proficiency (T1)	.00	1.00	-.52***	-.10	-.03	1.00							
5.Initial CT Proficiency (T2)	.00	1.00	-.24**	-.60***	.01	.32**	1.00						
6.Prior Programming Experience	.30	.46	-.18*	-.13	-.02	-.03	-.03	1.00					
7. Gender	.49	.50	-.14	.01	.02	.01	.03	.18*	1.00				
8. PELL	.49	.50	-.03	-.04	.01	.02	.06	.04	.28**	1.00			
9.CodeTutor Self-Efficacy	3.47	1.04	.04	.01	-.01	.08	.04	.08	.06	-.01	1.00		
10. Support Source: Friends	3.17	1.14	-.12	-.16	-.03	.02	.05	-.15	.10	.05	.14	1.00	
11. Support Source: Faculty	2.39	1.00	-.04	.07	-.02	-.17	-.19*	.19*	-.09	.18*	-.00	.05	1.00
12. Support Source: Private Tutor	1.93	.71	-.26**	-.24**	.10	.05	.04	.14	.25**	.07	.02	.04	.18*
Note: * p < .1; ** p < .05; *** p < .01													

5.4.1 Main Effects of Tutor Initiation on CT Proficiency Growth

Table 4 reports pre- to post-test *CT Proficiency Growth* for the Reactive and Proactive conditions across the T1 and T2 tasks. Students in both conditions improved significantly from pre-test to post-test on overall CT proficiency and on several CT dimensions, particularly pattern recognition, decomposition, and abstraction. Growth in algorithmic design was weaker, suggesting that this dimension may require more sustained practice to show comparable gains.

Table 4. CT Growth for Pseudocoding (T1) and Coding (T2) Tasks in Reactive and Proactive Modes				
	Mean (Reactive)	**Mean (Proactive)**	**Diff**	**p**
T1 CT Proficiency Growth	.14*** (.21)	.13*** (.27)	.01	.81
T1 Pattern Recognition Growth	.18*** (.31)	.16** (.40)	.02	.71
T1 Algorithmic Design Growth	.07 (.27)	.08* (.33)	-.01	.78
T1 Decomposition Growth	.14*** (.21)	.11*** (.24)	.03	.58
T1 Abstraction Growth	.17*** (.32)	.16** (.37)	.01	.88
T2 CT Proficiency Growth	.13*** (.24)	.12*** (.25)	.01	.83
T2 Pattern Recognition Growth	.21*** (.40)	.22*** (.38)	-.01	.95
T2 Algorithmic Design Growth	.11** (.28)	.03 (.25)	.08	.19
T2 Decomposition Growth	.12*** (.24)	.12** (.27)	.00	.94
T2 Abstraction Growth	.10** (.31)	.12** (.29)	-.02	.73

Despite these within-group improvements, there was no statistically significant difference in CT Proficiency Growth between the reactive and proactive conditions for either T1 or T2. The estimated differences are not statistically distinguishable from zero across all dimensions. This indicates that, on average, the two tutor initiation modes yield comparable improvements in CT proficiency.

5.4.2 Expertise-Contingent Effects of Tutor Initiation

The absence of an average treatment effect suggests that the impact of tutor initiation may depend on learner characteristics. We therefore test the directional moderation predicted in §2.4: the effect of tutor initiation

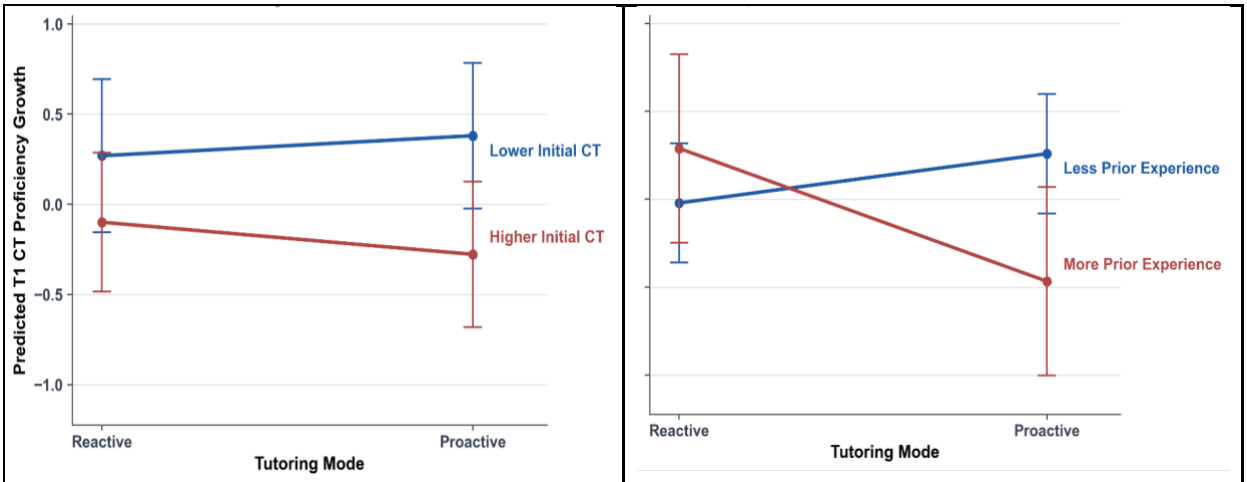
varies with learner expertise. Table 5 reports the results of moderation models estimating CT Proficiency Growth as a function of tutor initiation, Initial CT Proficiency, and their interaction, with a standard set of background and contextual controls included. The interaction between tutor initiation and Initial CT Proficiency is negative and for the CT Proficiency Growth in the conceptual reasoning task (T1) ($\beta = -.48$, $p < .05$), indicating the effect of proactive initiation declines as learners' initial CT proficiency increases.

Table 5. Results for CT Proficiency Growth

DV: CT Proficiency Growth	**Pseudocoding Task (T1)**	**Coding Task (T2)**
	Coef (S.E.)	**Coef (S.E.)**
TutorInitiationMode	.17 (.55)	.81 (.52)
InitialCTProficiency	-.23 (.15)	-.62 (.18)***
PriorProgrammingExperience	.18 (.34)	.13 (.35)
InitialCTProficiency*TutorInitiationMode	-.49 (.23)**	.25 (.24)
PriorProgrammingExperience*TutorInitiationMode	-1.05 (.48)**	-.31 (.45)
Gender	-.32 (.32)	.45 (.30)
Pell	-.16 (.30)	.24 (.28)
Ethnicity	.91(.66)	-.27 (.71)
Race.2	.69 (.36)*	.34(.36)
Race.3	-.25 (.54)	.61(.50)
Race.4	-1.60 (1.20)	-.51(1.25)
Race.5	-.16 (.58)	-.36(.55)
CourseSection.2	-.87 (.58)	-.12 (.54)
CourseSection.3	-.96 (.48)*	-.42 (.47)
CourseSection.4	-.15 (.39)	.12 (.38)
CodeTutorSelfEfficacy	.10 (.12)	-.05 (.11)
SupSources: friends	-.16 (.09)	-.10 (.09)
SupSources: faculty	.13 (.12)	.01 (.11)
SupSources: private tutor	-.49 (.17) ***	-.33 (.16)**
Gender*TutorInitiationMode	.78 (.43)*	-.18 (.40)
PELL*TutorInitiationMode	.18 (.44)	-.81 (.42)*
Ethnicity*TutorInitiationMode	-1.73 (.90)*	-.13 (.90)
Race*TutorInitiationMode.2	-.78 (.50)	.23 (.49)
Race*TutorInitiationMode.3	.68 (.79)	-.23 (.75)
Race*TutorInitiationMode.4	1.79 (1.56)	2.39 (1.56)
CourseSection*TutorInitiationMode.2	.56 (.77)	-.41 (.71)
CourseSection*TutorInitiationMode.3	.68 (.70)	.27 (.66)
CourseSection*TutorInitiationMode.4	-.35 (.59)	-.79 (.56)
Constant	1.49 (.62) **	.50 (.58)
Note: * p < .1; ** p < .05; *** p < .01		

Simple-slope analysis further clarifies this pattern. For learners with low Initial CT Proficiency, Proactive initiation is associated with greater CT Proficiency Growth in T1 ($dy/dx = .57$, $p < .10$). For learners with high Initial CT Proficiency (+1 SD), this advantage disappears and reverses direction ($dy/dx = -.40$, $p > .10$). These estimates imply that moving from reactive to proactive initiation is associated with a 0.57 unit increase in CT Proficiency Growth for lower-proficiency learners, but a 0.40 unit decrease for

higher-proficiency learners, indicating a meaningful divergence in learning outcomes across expertise levels. A similar expertise-contingent pattern emerges when prior programming experience is used as an alternative indicator of expertise. The interaction between tutor initiation and prior experience is negative and significant ($\beta = -1.01$, $p < .05$), with proactive initiation benefiting less experienced learners ($dy/dx = .42$, $p < .10$) but not more experienced learners ($dy/dx = -.59$, $p < .10$). The shift from reactive to proactive initiation corresponds to a 0.42 unit increase in CT Proficiency Growth for less experienced learners, but a 0.59 unit decrease for more experienced learners. Figure 4 illustrates these effects.



A. Moderated by Initial CT Proficiency | **B. Moderated by Prior Programming Experience**

Figure 4. Expertise-Contingent Effects of Proactive Tutoring on T1 CT Proficiency Growth

No comparable interaction effects are observed for the code implementation task (T2), consistent with the phase-contingent expectation (§2.6) that the expertise-contingent effects of tutor initiation concentrate in conceptual reasoning (where diagnostic responsibility is most acute). Interactions between Tutor Initiation Mode and the demographic and course-section covariates are included as an additional specification. The focal Initial CT × Mode and Prior Programming Experience × Mode interactions hold at $p < .05$ with these terms in the model.[3] These results suggest that the effect of tutor initiation depends on learner expertise. Proactive initiation is more effective for learners with lower initial proficiency or less experience, whereas reactive initiation yields better outcomes for more advanced learners.

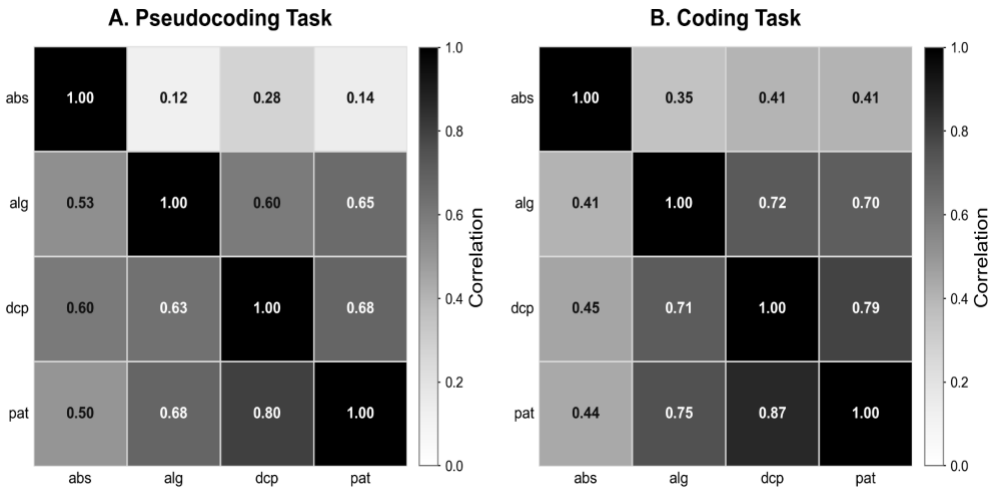
[3] Three of the coefficients reach marginal significance: Gender × Mode ($\beta = .78$, $p < .10$ [T1]), PELL × Mode ($\beta = -.81$, $p < .10$ [T2]), and Ethnicity × Mode ($\beta = -1.73$, $p < .10$ [T1]). These results are best read as exploratory.

5.4.3 Effects on Structural CT Integration

Beyond overall performance gains, we examine whether tutor initiation influences how learners coordinate multiple CT dimensions during problem-solving, as reflected in changes in Structural CT Integration. Table 6 reports changes in the covariance among CT dimensions from pre-test to post-test, and Figure 5 presents the full correlation structure among the four CT dimensions.

Table 6. Covariance Structure of CT Proficiency Dimensions

	T1			T2		
Metric	**Full**	**Reactive**	**Proactive**	**Full**	**Reactive**	**Proactive**
Avg inter-item r (pre)	.41	.45	.37	.56	.56	.60
Avg inter-item r (post)	.62	.61	.65	.60	.54	.68
Change in avg r (post - pre)	.21	.16	.28	.04	-.02	.09
Bootstrap 95% CI for change	[.04, .35]	[-.11, .37]	[.06, .45]	[-.13, .19]	[-.24, .19]	[-.11, .25]
Bootstrap two-sided p	.01	.23	.01	.70	.45	.83
Box's M: Chi-square	34.03	13.27	32.69	25.72	18.17	13.32
Box's M: df	10	10	10	10	10	10
Box's M: *p*	.00	.21	.00	.00	.05	.21



Note: abs: abstraction; alg: algorithmic design; dcp: decomposition; pat: pattern recognition

Figure 5. Correlations Between CT Dimensions in the Pre-Test and Post-Test

The proactive initiation in the conceptual reasoning task (T1) shows a larger increase in covariance among CT dimensions than reactive initiation, indicating stronger skill coupling after tutoring. Analysis reveals the biggest increases involve links between abstraction and decomposition and pattern recognition. These findings show that improvements in CT dimensions are linked rather than independent. No significant changes occur in the code implementation task (T2), aligning with phase-contingent expectations. Gains focus on conceptual reasoning. CT coupling is more adaptable. These results

demonstrate that tutor initiation influences the size and structure of learning gains. Even with similar overall improvements, differences exist in how learners coordinate CT dimensions during problem-solving.

6. PROCESS ANALYSIS OF TUTOR INITIATION MECHANISMS

The results reveal a systematic expertise-contingent effect of tutor initiation, in which the relative effectiveness of proactive and reactive support depends on learner expertise. Although both modes lead to comparable average improvements in CT proficiency, they differ in how they shape learners' engagement in the problem-solving process and the coordination of CT skills. These findings point to a central mechanism by which tutor initiation redistributes diagnostic responsibility between the learner and system.

To explain how these effects arise during interaction, we analyze fine-grained trace data from the tutoring sessions. These data allow us to explain how the observed expertise-contingent effects unfold during tutoring interaction, focusing on shifts in diagnostic responsibility and learner responses to support. We focus on three interaction activities (help articulation, error recovery, and solution navigation), each capturing a distinct aspect of the problem-solving process. Table 7 summarizes the focal constructs, underlying tensions, operational measures, and analytic approaches for each activity. Additional analysis and results are reported in Appendix F.

Table 7. Process Analysis: Activity-Level Constructs, Tensions, and Measures

Activity	Focal Construct	Underlying Tension	Operational Metric	Analytic Approach
Help Articulation	Diagnostic specificity in help articulation	Problem-Specific Articulation vs Confusion Expression	Diagnostic Specificity Index	Mixed-effects model; within-group change by mode × Initial CT Proficiency
Error Recovery	Resolution of sustained error states	Productive Struggle vs. Tutor-Assisted Recovery	Resolution Latency	Descriptive comparison of mean recovery times; wheeling-to-DSI comparison
Solution Navigation	Solution path inertia	Path Retention vs Structural switching	Structural Switching Rate; Processing Latency	CodeBERT trace alignment; latency comparison

6.1 Data and Measurement Overview

The tutoring session generated time-stamped chat logs and IDE traces spanning both the conceptual reasoning and code implementation phases. These traces link learner messages, tutor responses, proactive triggers, code submissions, and execution outcomes at the level of individual interaction episodes. Using these data, we constructed measures of diagnostic specificity in help articulation, resolution latency during sustained error states, and switching toward tutor-suggested solution paths. Consistent with prior work on

system use (Burton-Jones and Straub 2006), these measures capture how learners engage with the system during problem solving, rather than simply how frequently they use it.

6.2 Help Articulation

We examined how learners articulate their need for help during tutoring, focusing on diagnostic specificity (i.e., how well help requests identify specific difficulties rather than general confusion). Higher specificity indicates problem-specific requests, such as identifying errors or code segments. We used the Diagnostic Specificity Index (DSI), defined as the proportion of diagnostically specific messages among all requests. Each message was classified with a keyword-based NLP classifier. Messages were labeled diagnostically specific if they referenced errors, debugging, syntax issues, incorrect output, expected behavior, or Python errors. Otherwise, they were non-specific, indicating broader confusion.

To quantify the within-session shift in *diagnostic specificity*, a standardized mixed-effects model was estimated with DSI (z-scored) as the dependent variable, phase (conceptual reasoning = 0, code implementation = 1) as the primary predictor, and tutoring mode and initial CT level as covariates, with a random intercept per learner to account for baseline individual differences in help-articulation specificity.

The model revealed a significant phase effect ($\beta = .53$, SE = .26, $z = 1.99$, $p = .05$), indicating that diagnostic specificity increased as learners moved from planning to implementation. However, the absence of significant interaction effects between Tutor Initiation and Phase ($\beta = .03$, $p = .92$), as well as between the Initial CT Proficiency and Phase ($\beta = .29$, $p = .33$), suggests that the overall increase in *diagnostic specificity* across phases does not differ significantly by tutor initiation or learner expertise.

However, these results do not preclude systematic variation across learner groups. To examine this possibility, we analyze within-group changes in *diagnostic specificity* by tutor initiation and learner expertise (Table 8; Figure 6). These analyses reveal a clear pattern of expertise contingency. In the reactive condition, learners with higher Initial CT Proficiency showed a significant increase in diagnostic specificity (+15.2%, $p = .01$). In contrast, those with lower Initial CT Proficiency exhibited a modest and non-significant increase (+6.35%, $p = .15$). In contrast, under proactive initiation, learners with lower Initial CT

Proficiency showed a significant increase in diagnostic specificity (+10.5%, p = .00), whereas learners with higher Initial CT Proficiency exhibited only a marginal gain (+10.1%, p = .06).

Table 8. Change in Diagnostic Specificity from Planning to Implementation		
Group	**Δ Diagnostic Specificity**	***p***
Reactive / Higher Initial CT Proficiency	+15.2%	.01
Reactive / Lower Initial CT Proficiency	+6.35%	.15
Proactive / Higher Initial CT Proficiency	+10.1%	.06
Proactive / Lower Initial CT Proficiency	+10.5%	.00

These results indicate that tutor initiation redistributes diagnostic responsibility, thereby shaping how learners articulate their difficulties. Reactive initiation strengthens learner-generated diagnosis among more advanced learners by requiring them to independently identify and frame problems. Proactive initiation, by contrast, provides early diagnostic scaffolding that enables less experienced learners to formulate more specific, problem-focused help requests. We show the distributions in Figure 6.

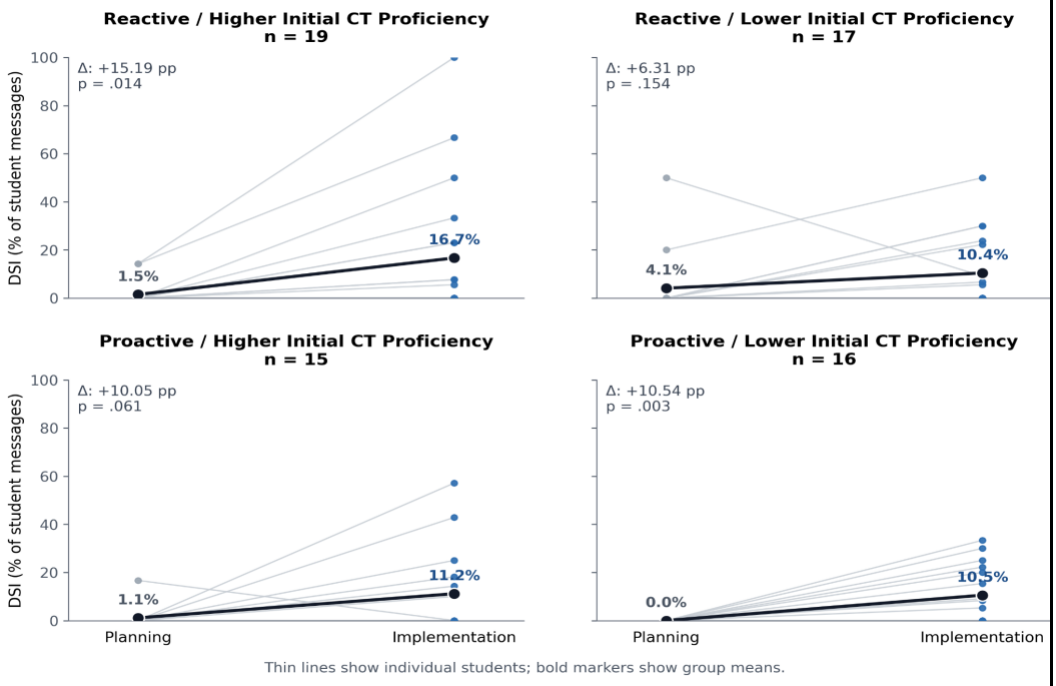


Figure 6. Distribution of DSI by Phase, Mode, and Initial CT Proficiency

Our findings demonstrate that tutor initiation shapes not only whether learners seek help but also how they articulate their difficulties. By shifting diagnostic responsibility between the learner and the tutor, proactive and reactive initiation modes produce distinct patterns of help articulation that vary systematically with learner expertise. Appendix F2 further decomposes help articulation into finer-grained

functional message categories, providing additional insight into how diagnostic specificity is expressed across different types of learner requests.

6.3 Error Recovery

We next examine how learners recover from sustained error states during code implementation. We define a sustained error state ("wheeling") as repeated failed execution with minimal code change, where minimal change is assessed using embedding-based similarity of consecutive submissions. A wheeling episode begins at the second consecutive such failure. This measure captures periods in which learners are unable to resolve errors independently. The focal construct, *resolution latency*, is defined as the time from the onset of a wheeling episode to the first subsequent successful execution.

To assess differences in recovery dynamics, we compare mean time-to-resolution across tutor initiation modes and levels of learner expertise (Table 9). Among learners with higher Initial CT Proficiency, those in the proactive condition recover substantially faster from wheeling episodes than those in the reactive condition (150 vs. 356 seconds, a difference of 206 seconds; $t = 2.40$, $p = .02$). In contrast, among learners with lower Initial CT Proficiency, proactive initiation is associated with a smaller and statistically insignificant reduction in recovery time (a difference of 69 seconds; $t = 0.49$, $p = .63$). These results indicate that proactive initiation accelerates error recovery primarily for more advanced learners, with weaker effects among less experienced learners. Figure F1 in Appendix F visualizes these recovery dynamics using Kaplan–Meier survival curves, and robustness checks using alternative measures of wheeling episodes yield consistent patterns (Appendix F3).

Beyond recovery speed, we also examine how sustained error states relate to subsequent help articulation. As Table 9 shows, learners who experience wheeling episodes subsequently exhibit higher diagnostic specificity than those who never enter such states, with a 15.9% higher mean DSI ($p < .001$). This relationship is pronounced in the reactive condition, where learners who experience wheeling episodes show a 20.3% increase in diagnostic specificity across phases ($p < .001$). These patterns suggest that extended engagement with unresolved error states is associated with more precise problem articulation, consistent with the role of productive struggle in facilitating deeper problem understanding (Kapur 2008).

Table 9. Error Recovery Outcomes and Associated Diagnostic Specificity			
Metric Group / Stratum	**Comparison Detail**	**Value**	***p***
Resolution Latency	Higher Initial CT Proficiency	-182.40 sec (HR = 2.24)	.02
	Lower Initial CT Proficiency	-48.71 sec (HR = 1.18)	.63
Diagnostic Specificity Gains	Episodes with vs. without wheeling	+15.9%	.00
	Wheeling episodes with phase shift	+20.3%	.00

These findings highlight a tension between productive struggle and tutor-assisted recovery. Proactive initiation reduces time spent in sustained error states by providing earlier guidance and improving efficiency in error resolution. However, this efficiency can limit opportunities for learner-generated diagnosis, particularly among advanced learners. Reactive initiation supports longer periods of independent problem-solving, during which learners refine their understanding of errors and develop more precise help articulation. Overall, tutor initiation shapes how quickly learners recover from errors and how they learn from them. By altering the balance between productive struggle and tutor-assisted support, proactive and reactive initiation modes produce distinct pathways for learners to develop diagnostic precision.

6.4 Solution Navigation

We last examine how learners navigate solution paths in response to tutor guidance. The focal construct is solution path inertia, reflected in the extent to which learners retain their current solution approach rather than engage in structural switching to explore alternatives. We conceptualize solution path inertia as a tension between path retention and structural switching, where *path retention* reflects incremental refinement while maintaining an existing solution approach and *structural switching* reflects substantive reorganization of the solution toward tutor-directed alternative approaches.

We operationalize solution path inertia using the Structural Switching Rate (SSR), which captures the proportion of transitions in which learners make substantial structural changes across consecutive submissions. Higher rates of structural switching indicate lower inertia, whereas greater path retention reflects higher inertia. Structural changes are identified using embedding-based code-similarity metrics, where lower similarity indicates a shift in solution strategy.

Specifically, we compare the semantic similarity of consecutive code submissions before and after tutor interventions using embedding-based representations. A structural switch is identified when a

learner's code both aligns with the tutor-provided solution (more than 5% of the submitted code overlaps with the tutor's solution) and diverges from the learner's prior implementation (i.e., similarity between pre- and post-intervention code falls below 0.85). Cases that do not meet these criteria are classified as path retention. Detailed classification procedures, threshold specifications, illustrative examples, and sensitivity analysis are provided in Appendix F4.

In addition to solution navigation patterns, we examine *processing latency*, defined as the time between a tutor intervention and the learner's next action. This measure captures the cognitive effort required to evaluate and integrate tutor guidance and complements the switching analysis by reflecting the temporal cost of responding to interventions. Because processing latency is tied to tutor interventions, this analysis focuses only on proactive intervention episodes.

Table 10 summarizes differences in structural switching and associated processing latency by tutor initiation and learner expertise. Among learners with higher Initial CT Proficiency, structural switching is rare (1.2%), indicating strong path retention even when unsolicited guidance is available. In contrast, among less experienced learners, switching occurs more frequently (13.5%), suggesting greater reliance on tutor-directed structural switching. This pattern is consistent with cognitive offloading, as learners may delegate aspects of solution structuring to the tutor during cognitively demanding phases such as planning and problem formulation (Fan et al. 2025).

Table 10. Structural Switching and Processing Latency in Proactive Intervention Episodes

Result	Value	*p*
Structural switching rate, Higher Initial CT Proficiency	1.2%	-
Structural switching rate, Lower Initial CT Proficiency	13.5%	-
Additional processing latency for structural switching vs path retention	+25.2 sec	.22
Increase in processing latency for novice learners after proactive nudges	+102%	< .01

Processing latency further differentiates structural switching and path retention. Structural switching is associated with longer response times than path retention (+25.2 seconds, $p = .22$), indicating that adopting tutor-suggested solution structures may require greater cognitive effort than refining an existing solution. However, the difference is not statistically significant. More notably, among less experienced learners, proactive tutor interventions are associated with a substantial increase in processing latency (+102%, $p < .01$), suggesting that integrating tutor-provided guidance imposes a measurable

cognitive burden when learners' own solution paths are still developing. These patterns indicate that structural switching is not frictionless but involves additional cognitive effort to interpret and integrate externally provided structure, consistent with prior work on how GenAI support shapes reflective processes during problem-solving (Abdel-Karim et al. 2023).

6.5 Process Synthesis: Linked Activity-Level Mechanisms

The process analysis reveals that tutor initiation operates through a set of linked activity-level mechanisms rather than a single undifferentiated effect. Across help articulation, error recovery, and solution navigation, proactive and reactive initiation shape different aspects of the interaction in ways that vary with learner expertise. These results indicate that the effectiveness of tutor initiation depends not on a universally superior mode but on how support is calibrated to learners' capabilities across stages of problem solving.

A central cross-activity linkage emerges between Error Recovery and Help Articulation. Time spent in sustained error states contributes to a more precise subsequent diagnosis. Learners who remain longer in self-directed wheeling episodes, particularly under reactive initiation, produce more diagnostically specific help requests in later stages. This pattern indicates that productive struggle is not merely a delay in progress but an active cognitive state that allows learners to extract diagnostic insight. Proactive tutoring prioritizes faster recovery, which can keep novices moving but may curtail diagnostic precision by assuming the framing burden for the student.

At the same time, proactive initiation reshapes the interaction in a different direction. By providing diagnostic scaffolding earlier, it reduces time spent in sustained error states, supports faster recovery, and increases exposure to tutor-directed solution paths. These features are especially beneficial for novice learners, who may lack the resources to diagnose problems, recover efficiently from repeated errors, or stabilize a coherent solution strategy. In this context, proactive initiation lowers barriers to continued engagement and supports progress through scaffolding.

Reactive initiation, by contrast, is more favorable when learners can use autonomy productively. By withholding unsolicited support, it preserves the need for learners to formulate their own diagnoses, persist through unresolved errors, and retain ownership over their solution paths. Although these demands

can be costly for novice learners, they appear to benefit more advanced learners, for whom self-directed struggle contributes to more precise diagnosis and stronger agency over the problem-solving process.

Collectively, the process analysis grounds the expertise-contingent effects observed in the main results in three activity-level patterns. Proactive initiation helps novice learners by providing diagnostic and directional scaffolding where their own resources are limited. Reactive initiation benefits more advanced learners because it preserves opportunities for productive struggle and learner-driven problem solving. The contribution of the process analysis is thus to show how differences in help articulation, error recovery, and solution navigation align with the broader pattern observed in the experiment. Table 11 summarizes these cross-activity mechanisms. Whether these activity-level patterns also surface as student-level pathways linking tutor initiation to CT Proficiency Growth is examined in §6.6.

The phase asymmetry observed in §5 is itself a process-level finding. Tutor initiation has its greatest leverage early in the problem-solving sequence, when learners are constructing the conceptual representation and external execution feedback is unavailable. By the implementation phase, the compiler and IDE absorb much of the diagnostic work the tutor would otherwise perform, attenuating the marginal value of tutor-initiation choices. The expertise-contingent effects observed in conceptual reasoning but not in code implementation reflect this leverage-point structure: tutor initiation governs diagnostic responsibility where it is most concentrated in the problem-solving sequence.

Table 11. Patterns from the Process Analysis

Activity	Focal Tension	Empirical Pattern	What Proactive Mode Does	What Reactive Mode Does	Contribution
Help Articulation	Learner-Generated vs. Tutor-Supported Diagnosis	Diagnostic specificity rises across phases, but gains differ by mode × initial proficiency	Scaffolds novice learners into more specific help-seeking	Forces more advanced learners to generate diagnostic precision	Proactive tracks specific help-seeking among novice learners; reactive tracks diagnostic precision among advanced learners
Error Recovery	Productive Persistence vs. Tutor-Assisted Recovery	Proactive mode shortens wheeling; longer wheeling predicts later gains in diagnostic specificity	Accelerates recovery and reduces time in sustained error states	Preserves longer, self-directed persistence that can sharpen later diagnosis	Connects faster recovery with the development of diagnostic precision
Solution Navigation	Path Retention vs Structural Switching	More switching and longer pauses among novice proactive learners; low switching among advanced learners	Exposes tutor-recommended paths and supports directed restructuring with a processing cost	Preserves self-conceived solution paths by limiting unsolicited directional cues	Surfaces the trade-off: proactive support scaffolds novices while reducing ownership over solution direction for some more advanced learners

6.6 Process-Outcome Linkage: Student-Level Analysis

To examine whether the activity-level process patterns also appear as student-level pathways linking tutor initiation to CT Proficiency Growth, we aggregated ΔDSI, mean resolution latency, and structural switching rate to the student level and estimated moderated mediation models. Tutor Initiation served as the independent variable. Each process measure was tested separately as the mediator. CT Proficiency Growth (T1) was the dependent variable. Initial CT Proficiency was the moderator. Continuous variables were standardized before estimation, confidence intervals were computed using 5,000 student-level bootstrap resamples, and students without an observed event for a given process stream were conservatively zero-coded. Table 12 reports the estimated conditional indirect effects for the three process measures.

Table 12. Student-Level Process–Outcome Linkage: Moderated Mediation Analysis Results

Task	Mediator	a: Tutor Initiation → Mediator		b: Mediator → CT Proficiency Growth	Conditional Indirect Effects		Difference (High − Low)
		Low Initial CT	High Initial CT		Low Initial CT	High Initial CT	
T1	ΔDSI	-0.15 [-0.68, 0.33]	-0.06 [-0.68, 0.52]	0.11 [-0.03, 0.28]	-0.02 [-0.10, 0.05]	-0.01 [-0.09, 0.08]	0.01 [-0.09, 0.13]
	Mean resolution latency	-0.54 [-1.49, 0.29]	0.25 [-0.33, 0.85]	0.26** [0.06, 0.38]	-0.14 [-0.38, 0.07]	0.06 [-0.07, 0.23]	0.20 [-0.10, 0.56]
	Structural Switching Rate	0.09 [-0.15, 0.36]	0.52 [0.00, 1.39]	0.07 [-0.02, 0.85]	0.01 [-0.01, 0.05]	0.04 [-0.01, 0.12]	0.03 [-0.01, 0.10]
T2	ΔDSI	-0.15 [-0.69, 0.36]	-0.06 [-0.68, 0.52]	-0.09 [-0.27, 0.22]	0.01 [-0.05, 0.14]	0.01 [-0.07, 0.12]	-0.01 [-0.14, 0.11]
	Mean resolution latency	-0.54 [-1.49, 0.29]	0.25 [-0.35, 0.86]	0.01 [-0.235,0.44]	-0.01 [-0.12, 0.11]	0.00 [-0.10, 0.05]	0.01 [-0.18, 0.16]
	Structural Switching Rate	0.09 [-0.14, 0.36]	0.52 [0.00, 1.38]	0.14 [-0.02, 0.31]	0.01 [-0.03, 0.03]	0.07 [-0.00, 0.26]	0.06 [-0.00, 0.27]

Some pathways, particularly mean resolution latency for T1 CT growth, are directionally aligned with the activity-level patterns reported in §6.2-§6.4. However, none of the conditional indirect effects or their high-minus-low differences are statistically significant at the 95% level, as all bootstrap confidence intervals include zero. These results should therefore be interpreted as suggestive evidence consistent with, but not formally establishing, a process mechanism linking tutor initiation to CT Proficiency Growth. This pattern likely reflects the aggregation of fine-grained interaction data to the student level, which attenuates observable relationships. Appendix F5 reports power calculations and simulation analyses assessing statistical sensitivity and directional stability. These analyses indicate that the directional patterns are likely to persist in larger samples, even though statistical significance is difficult to detect at the current sample size. Taken together, the student-level results are consistent with the finding that tutor initiation influences learning outcomes through the processes identified in §6.2–§6.5.

7. DISCUSSION AND IMPLICATIONS

The findings show that the value of tutor initiation is contingent on learner expertise rather than being uniformly beneficial. While both proactive and reactive initiation modes lead to comparable average improvements in CT proficiency, they shape the learning process in fundamentally different ways. Specifically, the findings form a Mode × Task contingency anchored in the feedback environment. In T1 (feedback-sparse), Mode interacts with learner expertise: proactive, favoring novices, reactive, favoring more advanced learners. In T2 (feedback-dense), the IDE absorbs the tutor's diagnostic role, and no Mode contingency emerges. The experiment identifies this expertise-contingent pattern, and the process analysis traces this pattern to differences in help articulation, error recovery, and solution navigation. Table 13 summarizes these integrated findings and their implications for theory, practice, and future research.

Table 13. Integrated Findings and Their Implications

Empirical finding	Process explanation	Implication for theory	Practical Implication	Future Research
No average treatment effect on CT Proficiency Growth (T1, T2)	Modes redistribute diagnostic responsibility differently; effects vary by expertise.	Heterogeneous treatment effects require AI tutor design to be theorized as expertise sensitive.	Avoid one-size-fits-all tutor behavior.	Test generalizability across domains, populations, and settings.
Proactive initiation benefited lower-proficiency learners	Earlier intervention supplied diagnostic framing, shortened error states, and guided solution navigation.	Tutor initiation compensates for limited self-diagnostic capacity.	Use proactive-first support for novices.	Identify thresholds where proactive support plateaus or over-scaffolds.
Reactive initiation benefited higher-proficiency learners	Learners generated more precise diagnoses, persisted longer, and retained solution ownership.	For advanced learners, autonomy is integral to the learning mechanism.	Use reactive-first support for advanced learners.	Distinguish productive struggle from inefficient cycling in real time.
Effects concentrate in conceptual reasoning, not code implementation (§5.4.2, §5.4.3)	Conceptual reasoning demands self-diagnosis; implementation provides compiler/IDE feedback that substitutes for tutor support.	Problem-solving phase moderates tutor-initiation via diagnostic responsibility distribution.	More proactive support in conceptual phases; restrain during implementation.	Test phase-adaptive designs across problem-solving stages and task environments.

7.1 Theoretical Implications

This study advances research on AI-supported learning and human–AI interaction by characterizing tutor initiation as a delegation configuration of an agentic IS artifact (Baird and Maruping 2021). Specifically, proactive initiation transfers diagnostic responsibility to the system, whereas reactive initiation retains it with the learner. While existing research focuses on the content and form of instructional support, our findings show that when and by whom help is initiated determines how diagnostic responsibility is allocated

between the learner and the system during problem-solving. We extend research on human–AI delegation (e.g., Fügener et al. 2022, Liu et al. 2025) from task settings with fixed objectives to learning environments in which delegation decisions recursively shape human capability. In this context, the learner acts both as the principal allocating diagnostic effort and as the agent whose competence evolves as a function of that allocation. Consequently, an LLM tutor is not merely a source of assistance but an agentic artifact whose initiation policy structures the cognitive division of labor underlying learning dynamics.

Second, the findings extend expertise-contingent learning theory to LLM tutoring. Prior work shows instructional support benefits novices but can be redundant or harmful for advanced learners (Chen et al. 2017), and the effectiveness of that guidance depends on alignment with cognitive traits (Krakowski et al. 2023, Wang et al. 2024). Our work demonstrates that the amount and timing of support matter, especially whether the system intervenes before learners diagnose difficulty. The same initiation mode can boost CT Proficiency for one group but hinder another, as AI helps improve performance but may reduce engagement in reasoning (Li et al. 2026). Thus, the value of the system initiation depends on the learner's expertise in handling diagnostic responsibilities. Our study refines theory by showing how tutor initiation influences expertise-contingent effects and how learner expertise affects interaction with AI assistance.

Third, the study explores how AI-supported learning unfolds through interaction rather than solely through outcomes. Learning emerges from interaction dynamics (Gupta and Bostrom 2013, Wambsganss et al. 2025). We identify specific processes like Help Articulation, Error Recovery, and Solution Navigation, which influence how tutor initiation affects outcomes. These effects depend on how learner autonomy and tutor scaffolding are enacted across activities, not fixed properties. Models should consider how system initiation shapes different problem-solving activities rather than assuming uniform effects.

Fourth, the study identifies the problem-solving phase as a moderator of tutor-initiation effects, alongside learner expertise. Tutor initiation is most consequential during conceptual reasoning, when learners must diagnose problems and evaluate solutions. During code implementation, compiler errors and runtime exceptions partially substitute for the tutor's diagnostic role, reducing the marginal impact of initiation. By introducing phase as a moderator, our work refines theoretical models of AI-supported

learning: the effectiveness of system support depends not only on learner expertise and the form of guidance provided, but also on how cognitive demands are distributed across the task environment.

7.2 Practical Implications for Design and Pedagogy

The findings have important implications for the design of AI tutoring systems. The results provide proof of value for situated implementations of AI tutoring systems (Nunamaker et al. 2015), demonstrating that their effectiveness depends not only on the availability of support but on how tutor initiation is calibrated. Rather than assuming that more tutor initiation is always beneficial, designers must treat it as a parameter that shapes how learning unfolds. This reflects a well-known assistance dilemma (Koedinger and Aleven 2007), in which both insufficient and excessive support can hinder learning.

From a design perspective, the central implication is to calibrate the allocation of diagnostic responsibility between the learner and the system. Proactive intervention can provide essential diagnostic framing, reduce time spent in unresolved error states, and guide learners through solution paths when they are unable to do so independently. In contrast, reactive intervention preserves opportunities for self-directed diagnosis, productive struggle, and ownership over the evolving solution. The effectiveness of each approach depends on whether learners can carry diagnostic responsibility productively.

These findings suggest several design principles for expertise-contingent AI tutoring. *First,* tutor initiation should vary in timing and intensity to support learners lacking diagnostic capacity while avoiding displacing self-generated reasoning among advanced learners. *Second,* systems should preserve opportunities for learner-driven diagnosis, ensuring support does not replace diagnostic development. *Third,* proactive intervention can guide less experienced learners toward better solutions, especially during conceptual reasoning, where exposure to tutor-suggested paths helps. Designers should balance efficiency from reduced error states against limiting productive struggle. Proactive intervention benefits advanced learners by refining without shifting solutions significantly, whereas reactive strategies can hinder less experienced learners by prolonging unproductive struggle. Fourth, tutor initiation should match feedback availability: proactive is useful with sparse feedback, such as early problem-solving or pseudocoding, whereas reactive is better with dense feedback, like later phases or debugging.

AI tutoring should be integrated as a structured support aligned with how students learn at different levels, not just as a convenience for answering questions (Roschelle et al. 2025). Early on, proactive support can help organize ideas and reduce confusion, while during coding, a restrained approach fosters learner responsibility. Pedagogy must be learner- and phase-sensitive, especially in MSIs with students from varied backgrounds. Adaptive tutor initiation can differentiate support without forcing uniform intervention. The key is to align tutor activity with the learner's capacity for productive problem-solving.

7.3 Future Research Implications

The study opens several directions for research on AI-supported learning. First, the findings should be tested beyond introductory programming and the MSI context. Although programming is a strong setting for observing computational thinking development, the logic of tutor initiation may extend to other domains in which learners must diagnose difficulty, persist through breakdowns, and decide whether to adopt externally suggested solution paths. Future work should examine whether these expertise-contingent effects generalize across tasks, disciplines, and learner populations.

Second, research should move beyond fixed tutoring to test adaptive tutor initiation. Because the value of initiation varies by learner expertise and problem-solving phase, tutors should be designed to adjust initiation dynamically in response to signals such as low diagnostic specificity, prolonged unresolved error states, or repeated reliance on tutor-suggested paths. This would shift research from documenting contingent effects to identifying when initiation should recede, intensify, or change form.

Third, research should identify the causal mechanisms underlying the differences in help articulation, error recovery, and solution navigation. Further research is needed to isolate which features of tutor initiation drive them. Experiments that vary intervention timing, guidance framing, or structural support could clarify which aspects matter most for learners at different expertise levels.

Fourth, research should test whether this phase-contingent logic generalizes across tasks that differ in execution feedback. Tutor initiation appears most consequential during conceptual reasoning, when learners must build a problem representation without execution feedback. It should therefore matter most

in tasks with sparse feedback, such as pseudocoding, algorithm specification, and system design, and least in tasks with dense feedback, such as debugging with a test suite or test-driven development.

Finally, future work should improve the measurement of learning processes in AI-supported environments. Precise methods are needed to capture productive struggle, diagnostic specificity, and structural switching; distinguish productive from unproductive struggle; determine when guidance supports learning rather than displacing learner agency; and track conceptual integration across problem-solving stages. These advances are essential for advancing theory and designing more adaptive AI tutoring systems.

8. CONCLUSION

The value of AI tutoring depends not only on whether support is available but also on when it enters the interaction and for whom. In a controlled study of an LLM-based tutor in introductory programming, we found no universal advantage of proactive over reactive tutoring. Instead, the effectiveness of tutor initiation is expertise-contingent: proactive initiation better supports learners with lower initial CT proficiency or less prior programming experience, whereas reactive initiation better supports more advanced learners. These effects concentrate on conceptual reasoning, where the tutor's diagnostic role is most consequential; once execution begins, the IDE and compiler take over much of the diagnostic work. Process analysis traced this pattern to how the two modes differentially redistribute diagnostic responsibility across help articulation, error recovery, and solution navigation. By identifying tutor initiation as a mechanism for allocating diagnostic responsibility, the study advances theory on AI-supported learning and provides a process-level account of expertise- and phase-contingent effects.

These insights have practical implications for educators and AI tutoring system designers, especially in settings characterized by heterogeneous learner expertise. Rather than adopting one-size-fits-all support, educators and designers should tailor tutor initiation to learners' diagnostic capabilities and to where in the problem-solving sequence diagnostic responsibility is concentrated. The promise of AI in education lies in structuring interaction to support learner reasoning.

REFERENCES

Abdel-Karim BM, Pfeuffer N, Carl KV, Hinz O (2023) How AI-based systems can induce reflections: The case of AI-augmented diagnostic work. *MIS Quart.* 47(4):1395–1424.

Aleven V, McLaren B, Roll I, Koedinger K (2006) Toward meta-cognitive tutoring: A model of help seeking with a Cognitive Tutor. *Internat. J. Artificial Intelligence Ed.* 16(2):101–128.

Aleven V, Roll I, McLaren BM, Koedinger KR (2016) Help helps, but only so much: Research on help seeking with intelligent tutoring systems. *Internat. J. Artificial Intelligence Ed.* 26(1):205–223.

Baird A, Maruping LM (2021) The next generation of research on IS use: A theoretical framework of delegation to and from agentic IS artifacts. *MIS Quart.* 45(1):315–341.

Bhat S, Varma V (2023) Large language models as annotators: A preliminary evaluation for annotating low-resource language content. *Proc. 4th Workshop Eval. Comparison NLP Systems*, 100–107.

Breazeal C, Rai A, Ramesh B, Chen L, Long Y, Aria A, Loi H, Torralba A, Bernstein J, Reich J, Klopfer E, Abelson H (2024) Opportunities, issues, and challenges for generative AI in fostering equitable pathways in computing education. Report, MIT Exploration of Generative AI, Cambridge, MA.

Brennan K, Resnick M (2012) New frameworks for studying and assessing the development of computational thinking. Presentation, *Annual Meeting of the American Educational Research Association*, Vancouver, BC. https://scratched.gse.harvard.edu/ct/files/AERA2012.pdf

Burton-Jones A, Straub DW (2006) Reconceptualizing system usage: An approach and empirical test. *Inform. Systems Res.*17(3):228–246.

Chen O, Kalyuga S, Sweller J (2017) The expertise reversal effect is a variant of the more general element interactivity effect. *Educational Psych. Rev.* 29(2):393–405.

Dever DA, Wiedbusch MD, Romero SM, Azevedo R (2024) Investigating pedagogical agents' scaffolding of self-regulated learning in relation to learners' subgoals. *British J. Educational Tech.* 55(4):1290.

Fan Y, Tang L, Le H, Shen K, Tan S, Zhao Y, Shen Y, Li X, Gašević D (2025) Beware of metacognitive laziness: Effects of generative artificial intelligence on learning motivation, processes, and performance. *British J. Educational Tech.* 56(2):489–530.

Fügener A, Grahl J, Gupta A, Ketter W (2022) Cognitive challenges in human-artificial intelligence collaboration: Investigating the path toward productive delegation. *Inform. Systems Res.* 33(2):678.

Gao Y, Xiong Y, Gao X, Jia K, Pan J, Bi Y, Dai Y, Sun J, Wang M, Wang H (2023) Retrieval-augmented generation for large language models: A survey. arXiv:2312.10997.

Gilardi F, Alizadeh M, Kubli M (2023) ChatGPT outperforms crowd workers for text-annotation tasks. *Proc. Natl. Acad. Sci.*120(30):e2305016120.

Gupta S, Bostrom R (2013) An investigation of the appropriation of technology-mediated training methods incorporating enactive and collaborative learning. *Inform. Systems Res.* 24(2):454–469.

Hwang GJ, Wang SY, Lai CL (2021) Effects of a social regulation-based online learning framework on students' learning achievements and behaviors in mathematics. *Comput. Ed.* 160:104031.

Johnson DR, Soldner M, Leonard JB, Alvarez P, Inkelas KK, Rowan-Kenyon HT, Longerbeam SD (2007) Examining sense of belonging among first-year undergraduates from different racial/ethnic groups. *J. College Student Development* 48(5):525–542.

Kalyuga S, Ayres P, Chandler P, Sweller J (2003) The expertise reversal effect. *Educ. Psychol.* 38(1):23-31.

Kapur M (2008) Productive failure. *Cogn. Instr.* 26(3):379–424.

Kim D, Mitrofanov D, Wen Q, Xu T (2025) Generative AI can improve performance and engagement without harming learning. Working paper, https://papers.ssrn.com/sol3/papers.cfm?abstract_id= 5929576.

Koedinger KR, Aleven V (2007) Exploring the assistance dilemma in experiments with cognitive tutors. *Educational Psych. Rev.*19(3):239–264.

Kosmyna N, Hauptmann E, Yuan YT, Situ J, Liao XH, Beresnitzky AV, Braunstein I, Maes P (2025) Your brain on ChatGPT: Accumulation of cognitive debt when using an AI assistant for essay writing task. arXiv:2506.08872.

Krakowski S, Luger J, Raisch S (2023) Artificial intelligence and the changing sources of competitive advantage. *Strategic Management J.* 44(6):1425–1452.

Li AT, Liu D, Xu SX, Yi C (2024) Interleaved design for e-learning: Theory, design, and empirical findings. *MIS Quart.*48(4):1363–1394.

Li AT, Liu D, Ye T (2026) Is ChatGPT a boon or a bane for learning? Experimental evidence across task formats and chatbot designs. Working paper, https://ssrn.com/abstract=5533921.

Lo CK, Hew KF, Jong MSY (2024) The influence of ChatGPT on student engagement: A systematic review and future research agenda. *Comput. Ed.* 219:105100.

Liu, J., Yue, W. T., Leung, A. C. M., & Zhang, X. (2025). Find the good. Seek the unity: A hidden Markov model of human-AI delegation dynamics. *MIS Quart.* 49(3):1185–1204.

MacKenzie SB, Podsakoff PM, Podsakoff NP (2011) Construct measurement and validation procedures in MIS and behavioral research: Integrating new and existing techniques. *MIS Quart.* 35(2):293–334.

Madaan A, Tandon N, Gupta P, Hallinan S, Gao L, Wiegreffe S, Clark P (2023) Self-refine: Iterative refinement with self-feedback. arXiv preprint arXiv:2303.17651.

Muralidharan K, Singh A, Ganimian AJ (2019) Disrupting education? Experimental evidence on technology-aided instruction in India. *Amer. Econom. Rev.* 109(4):1426–1460.

Nelson-Le Gall S (1981) Help-seeking: An understudied problem-solving skill in children. *Developmental Rev.* 1(3):224–246.

Nunamaker JF Jr, Briggs RO, Derrick DC, Schwabe G (2015) The last research mile: Achieving both rigor and relevance in information systems research. *J. Management Inform. Systems* 32(3):10–47.

Piccoli G, Ahmad R, Ives B (2001) Web-based virtual learning environments: A research framework and a preliminary assessment of effectiveness in basic IT skills training. *MIS Quart.* 25(4):401–426.

Polya G (1957) *How to Solve It: A New Aspect of Mathematical Method*, Princeton University Press.

Roschelle J, McLaughlin EA, Koedinger KR (2025) Beyond benchmarks: Responsible AI in education needs learning sciences. *Comm. ACM* 68(12):37–39.

Santhanam R, Sasidharan S, Webster J (2008) Using self-regulatory learning to enhance e-learning-based information technology training. *Inform. Systems Res.* 19(1):26–47.

Todd P, Benbasat I (1992) The use of information in decision making: An experimental investigation of

the impact of computer-based decision aids. *MIS Quart.* 16(3):373–393.

Shute VJ (2008) Focus on formative feedback. *Rev. Educational Res.* 78(1):153–189.

Stelmaszak M, Möhlmann M, Sørensen C (2025) When algorithms delegate to humans: Exploring human-algorithm interaction at Uber. *MIS Quart.* 49(1):305–330.

Straub D, Boudreau MC, Gefen D (2004) Validation guidelines for IS positivist research. *Comm. Assoc. Inform. Systems*13(1):24.

Tang X, Yin Y, Lin Q, Hadad R, Zhai X (2020) Assessing computational thinking: A systematic review of empirical studies. *Comput. Ed.* 148:103798.

Urban M, Děchtěrenko F, Lukavský J, Hrabalová V, Svacha F, Brom C, Urban K (2024) ChatGPT improves creative problem-solving performance in university students: An experimental study. *Comput. Ed.* 215:105031.

Van Dusen B, Nissen J (2020) Associations between learning assistants, passing introductory physics, and equity: A quantitative critical race theory investigation. *Phys. Rev. Phys. Ed. Res.* 16(1):010117.

VanLehn K (2011) The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educ. Psychol.* 46(4):197–221.

Von Zahn M, Liebich L, Jussupow E, Hinz O, Bauer K (2025) Knowing (not) to know: Explainable artificial intelligence and human metacognition. *Inform. Systems Res.* Forthcoming.

Walton GM, Cohen GL (2007) A question of belonging: Race, social fit, and achievement. *J. Personality Social Psych.*92(1):82–96.

Wambsganss T, Janson A, Söllner M, Koedinger K, Leimeister JM (2025) Improving students' argumentation skills using dynamic machine-learning–based modeling. *Inform. Systems Res.* 36(1): 474–507.

Wang W, Gao G, Agarwal R (2024) Friend or foe? Teaming between artificial intelligence and workers with variation in experience. *Management Sci.* 70(9):5753–5775.

Winkler R, Söllner M, Leimeister JM (2021) Enhancing problem-solving skills with smart personal assistant technology. *Comput. Ed.* 165:10414.

Calibrating LLM Tutor Initiation to Learner Expertise: Proactive and Reactive Support for Computational Thinking

ELECTRONIC COMPANION

APPENDIX A: FOCUS GROUP PROTOCOL AND DISCOVERY PROCEDURE

We conducted eight focus groups to understand student's learning needs (details in Table A1). For these focus groups, we recruited students from an introductory Python course at a large urban MSI. Participants joined the focus groups through voluntary sign-up based on their availability. The resulting participant pool reflected substantial variation in prior programming exposure (with or without programming experience before taking this course), academic discipline (IS, computer science, non-STEM majors), and socioeconomic context (more than half were Pell-eligible or first-generation students). Each 90-minute comparative think-aloud session had three stages:

- **Baseline struggle (15 min)**: Students solved a logic-intensive programming problem without AI support.
- **Comparative rotation (45 min)**: Students interacted with an early CodeTutor prototype alongside existing GenAI tutoring systems to directly compare support interactions across systems.
- **Open discussion and reflection** (**30 min**): Students participated in a moderated discussion about their experiences with different systems, including what they liked most, disliked most, perceived as helpful or unhelpful, and the types of support they wished the systems could better provide. These sessions generated rich transcripts and interaction logs.

Table A1. Focus Groups

	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7	Group 8
Time	April 1, 12:00 pm	April 1, 4:30 pm	April 3, 12:00 pm	April 3, 4:30 pm	April 18, 4:30 pm	April 19, 4:30 pm	July 18, 4:30 pm	July 19, 4:30 pm
Task	Concept comprehension	Debug	Exam preparation	Program writing	Debug	Concept comprehension	Program writing	Exam preparation
Topic	Dictionary	Loops and strings	Loops	Loops	Loops and strings	Dictionary	Loops	Loops
Valid N*	11	16	12	21	30*	30*	44*	35*
Tutoring Systems in Comparison	CS50 vs. Khanmigo	CS50 vs. Khanmigo	CS50 vs. Khanmigo	CS50 vs. Khanmigo	CodeTutor vs. CS50/Khanmigo	CodeTutor vs. CS50/Khanmigo	CodeTutor vs. CS50/Khanmigo	CodeTutor vs. CS50/Khanmigo

* 24 students attended two groups among Group 5-8.

Discussion Themes and Emergent Insights . Semi-structured discussion questions focused on three dimensions: (1) learner agency in identifying difficulties and articulating help requests; (2) interaction experience when students initiated support requests during problem-solving, ; and (3) curricular alignment betweenAI assistance and instructor expectations . Across these sessions, students consistently reported difficulty recognizing when they needed help and formulating effective questions during programming impasses. At the same time, they valued opportunities to retain control over the problem-solving process rather than receiving immediate solutions. These observations suggested that learners varied substantially in their ability to self-diagnose difficulties and strategically seek assistance during problem-solving.

The findings informed the design requirements for CodeTutor: the system should support learners who struggle to recognize or articulate breakdowns in reasoning while preserving learner control over problem-solving Appendix Figure A1 presents the full system architecture, highlighting a unified code-editor and chat interface, where telemetry monitors activity, a scaffolding layer governs support delivery, and a retrieval-augmented generation pipeline grounds responses in instructor-provided materials.

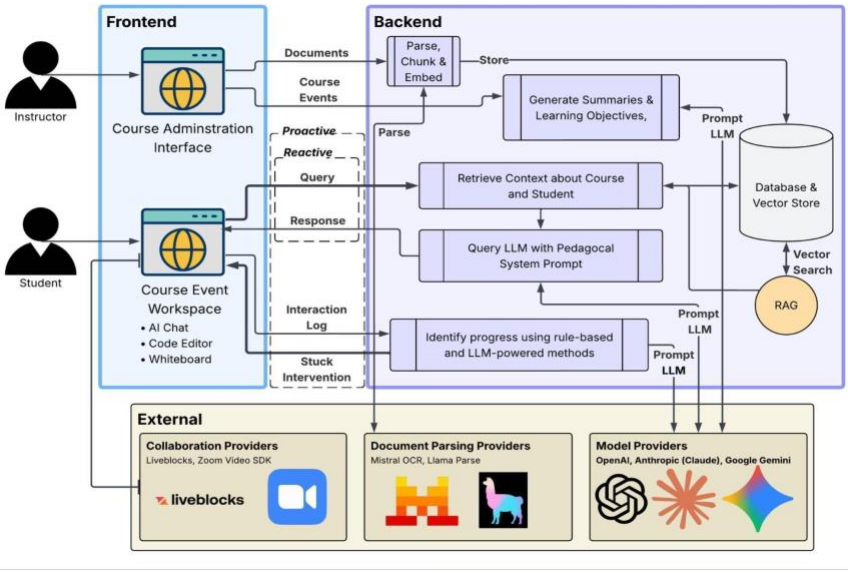


Figure A1. CodeTutor Architecture

APPENDIX B: CT PROFICIENCY ASSESSMENT RUBRIC

The rubric is grounded in contemporary conceptualizations of CT, integrating Wing's (2006) foundational definition with Brennan and Resnick's (2012) three-dimensional framework of computational concepts, practices, and perspectives, and Grover and Pea's (2013) state-of-the-field synthesis. The four dimensions assessed (i.e., decomposition, abstraction, pattern recognition, and algorithm design) align with the computational concepts dimension (Brennan and Resnick 2012) and with validated CT constructs underlying assessment instruments such as the Computational Thinking Test (Román-González et al. 2017). Consistent with prior empirical work, we adopt rubric-based scoring of programming artifacts and written reasoning, which is among the most widely used approaches for assessing CT proficiency (Tang et al. 2020). Given that many established CT instruments target K-12 populations, we adapt task design for undergraduate-level algorithmic reasoning, following recent work in higher education contexts (e.g., Lafuente Martínez et al. 2022). Table B1 provides the detailed rubric used to score CT proficiency. The rubric applies binary scoring (0 = absent, 1 = present) based on observable evidence in student submissions. It is applied consistently to the conceptual reasoning and code implementation tasks across the three phases (i.e., pre-test, interaction, and post-test).

Table B1. CT Proficiency Assessment Rubric

CT Dimension	Rubric Element	Evidence Criteria (Decision Rules for Scoring)
Decomposition	Input Handling	Specify what is being collected (e.g., "get names and grades")
	Data Structure	Identifies the data structure to use for the input process.
	Control Structure	Identifies the control structure to use for the input process.
	Sub-goal Identification	Separates the "collection phase" from the "calculation phase."
	Output Plan	Explicitly mentions displaying the result of the logic.
Algorithm Design	Sequencing	Logic follows a linear executable flow (a → b → c).
	Conditionals	Includes a "comparison" step (e.g., "if current > max").
	Iteration	Indicates a repeating process (e.g., "repeat for each student").
	State Updates	Mentions updating a variable (e.g., "store new highest grade").
	Termination	Specifies the conditions when the program should end.
Pattern Recognition	Input Pattern	Recognizes that data comes in pairs (e.g., "split string by spaces to get name and score").
	Aggregation Pattern	Recognizes the need for a "running" maximum or average calculation.
	Repeat Patterns	Identifies repeat patterns essential to traversal.
	Output Patterns	Identifies the pattern to display output.
Abstraction	Data Abstraction	Uses a structured format to link related data (e.g., "store as a list of pairs").
	Functional Abstraction	Identifies logic that can be encapsulated (e.g., "a helper to validate scores").

References:

Brennan K, Resnick M (2012) New frameworks for studying and assessing the development of computational thinking. Presentation, *AERA Annual Meeting*, Vancouver, BC.

Román-González M, Pérez-González JC, Jiménez-Fernández C (2017) Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Comput. Hum. Behav.* 72: 678–691.

Tang X, Yin Y, Lin Q, Hadad R, Zhai X (2020). Assessing computational thinking: A systematic review of empirical studies. *Comput. Ed.* 148:103798.

Lafuente Martínez M, Lévêque O, Benítez I, Hardebolle C, Zufferey JD (2022) Assessing computational thinking: Development and validation of the Algorithmic Thinking Test for adults. *J. Educ. Comput. Res.* 60(6): 1436–1463.

APPENDIX C: AUTOMATED SCORING PIPELINE AND PROMPT ENGINEERING

This appendix outlines the automated scoring and validation procedure used to apply the CT proficiency rubric across conceptual reasoning (T1) and code implementation (T2) tasks.

Scoring Framework. Implemented using GPT-4.1 mini with a temperature setting of 0.0 to ensure deterministic, reproducible outputs. Structured prompts required explicit justification for each scoring decision, and the model was restricted to evaluating only observable evidence in student submissions. This design ensured consistency in scoring and prevented inferences beyond the provided responses.

Cross-Format Standardization. To ensure comparability across task formats, code implementation responses (T2) were translated into ordered logical steps before evaluation. These steps were then assessed using the same rubric applied to conceptual reasoning responses (T1). This transformation ensured that scores reflect underlying computational thinking (CT) rather than differences in response format.

Prompt Design and Refinement. During calibration, initial prompts were evaluated to identify cases where vague or incomplete responses were incorrectly credited. In the refinement stage, prompt instructions were revised to enforce stricter evidence requirements and clarify the distinction between descriptive statements and procedural logic. In the validation stage, automated scores were compared with independent human ratings to assess consistency.

Validation Against Human Raters. Automated scores generated by a GPT-based grading pipeline were validated against ratings provided by five graduate research assistants (GRAs). Each rubric component was assigned a GRA responsible for evaluation across both the pre-test and post-test. For the pre-test, each component was evaluated by a second independent GRA drawn from the same pool, ensuring that all components received two independent ratings, enabling the assessment of inter-rater reliability. Cohen's kappa between the two human raters indicated strong agreement. This is illustrated by the decomposition dimension, which achieved $\kappa = 0.85$. For the post-test, each rubric component was evaluated by a single GRA.

To assess alignment between the GPT-based grading pipeline and human ratings, we compared model-generated ratings with human ratings using precision, recall, and F1 scores. For the pre-test, comparisons were conducted using the primary human rater's ratings. As shown in Table C1, the pipeline demonstrates strong classification performance, with F1 scores ranging from 0.80 to 0.96 across rubric components, alongside consistently high precision and recall.

Table C1. Accuracy of GPT results for grading decomposition

Decomposition grading rubric	Pre-test			Post-test		
	Precision	Recall	F1	Precision	Recall	F1
C1	82.7%	100.0%	0.905	84.1%	100.0%	0.914
C2	96.5%	87.3%	0.917	97.1%	94.4%	0.958
C3	83.3%	76.9%	0.800	93.5%	91.1%	0.923
C4	84.4%	98.5%	0.909	81.0%	98.5%	0.889
C5	90.0%	88.7%	0.894	90.4%	95.7%	0.930

Score Construction. Each CT dimension was scored independently using binary criteria. Scores were normalized by dimension to ensure comparability across tasks. An overall CT proficiency score was computed as the unweighted mean of the four dimensions.

APPENDIX D: MEASUREMENT ITEMS AND CONSTRUCT STATISTICS

Table D1 reports measurement items, scale definitions, and descriptive statistics for Prior Programming Experience, CodeTutor Self-Efficacy, and Support Sources.

Table D1. Measurement Items

Construct	Measure	Mean	SD	N
Prior Programming Experience	Please choose the programming language(s) you have learned before taking this course. Please select all that apply (None; C; C++; Java; JavaScript; Others). *Scale: 0= "None" is selected, 1 = all other options are selected*	.30	.46	90
CodeTutor Self- Efficacy	Please indicate the degree of your agreement with each item. - I felt comfortable using CodeTutor on my own to learn programming. - I could easily interact with CodeTutor on my own to learn programming. - I felt comfortable using CodeTutor to learn programming, even when no one was around to tell me how to use it. *Scale: 1= Strongly disagree to 5= Strongly agree*	3.47	1.04	85
Support Sources	During the current semester, how often do you seek help for any academic-related questions from the following sources? *Scale: 1=Never to 5 =Always*			
	- friends	3.17	1.14	87
	- faculty	2.39	1.00	87
	- private tutors	1.93	.71	87

APPENDIX E: SUPPLEMENTAL STRUCTURAL INTEGRATION ANALYSIS

E1. Pairwise Dimensional Shifts

Table E1 reports pairwise correlations among CT dimensions in the pre- and post-tests across the full sample, and separately for the reactive and proactive conditions. Changes in correlations (Δr) indicate shifts in the relationships among CT dimensions across phases. Δr is computed as the difference between post-test and pre-test correlations ($r_post - r_pre$). Increases are most pronounced in correlations involving abstraction during the pseudocoding task (T1).

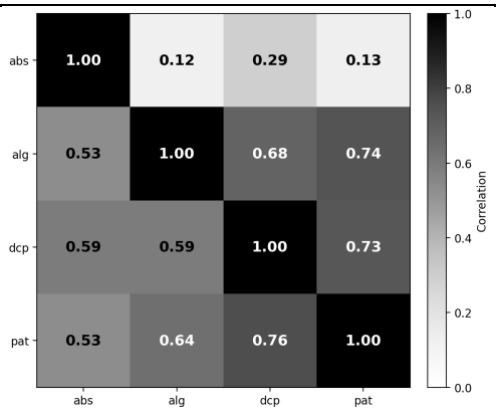
Table E1. Pairwise Comparison of Pre- and Post- Test Correlations Among CT Dimensions

Task	Pair	Full Sample				Reactive		Proactive	
		r_pre	r_post	Δr	p	Δr	p	Δr	p
T1	abs-alg	.13	.53	.40	.00	.42	.04	.40	.03
	abs-dcp	.28	.60	.32	.02	.31	.19	.35	.02
	abs-pat	.14	.50	.36	.01	.40	.07	.32	.12
	dcp-alg	.60	.63	.03	.65	.09	.34	.21	.16
	dcp-pat	.68	.80	.12	.09	.03	.83	.22	.02
	alg-pat	.65	.68	.03	.81	.11	.22	.18	.22
T2	abs-alg	.35	.41	.05	.74	.11	.57	.18	.25
	abs-dcp	.41	.45	.05	.72	.05	.74	.12	.50
	abs-pat	.41	.44	.03	.80	.10	.55	.17	.22
	dcp-alg	.72	.71	.01	.89	.02	.81	.00	.97
	dcp-pat	.79	.87	.08	.06	.11	.07	.06	.24
	alg-pat	.71	.75	.05	.45	.09	.28	.00	.99

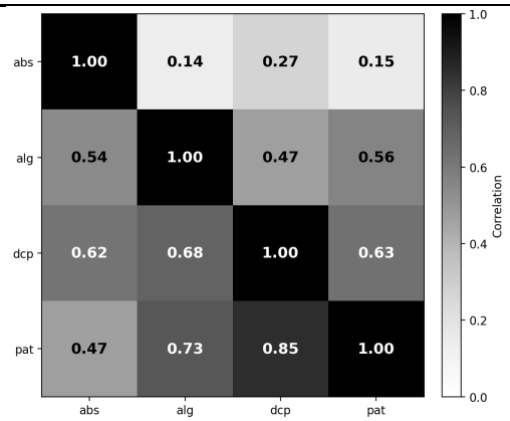
Note: abs: abstraction; alg: algorithmic design; dcp: decomposition; pat: pattern recognition

E2. Condition-Specific Heatmaps

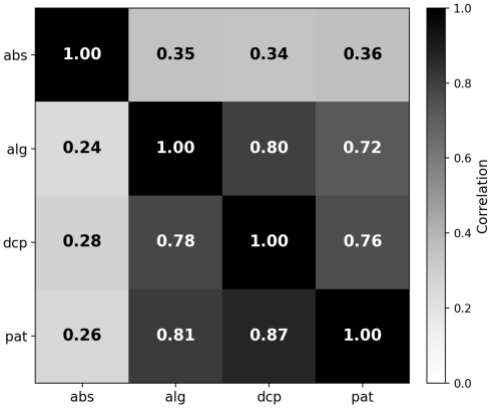
Figure E1 visualizes correlation patterns among CT dimensions in the pre- and post-tests across tasks and conditions.



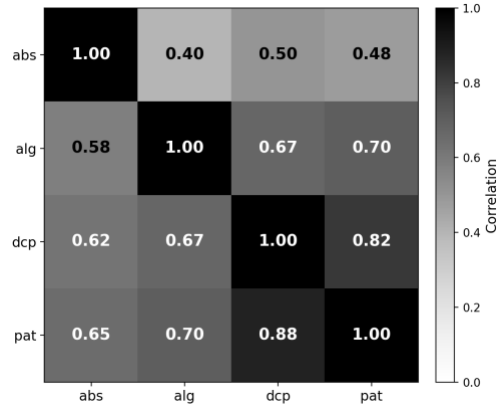
E1a. Pseudocoding Task in Reactive Mode



E1b. Pseudocoding Task in Proactive Mode



E1c. Coding Task in Reactive Mode



E1d. Coding Task in Proactive Mode

Note: abs: abstraction; alg: algorithmic design; dcp: decomposition; pat: pattern recognition

Figure E1. Correlation Between CT Dimensions in the Pre-Test and Post-Test

APPENDIX F: PROCESS ANALYSIS

This appendix describes the process-level measures used to analyze student–system interactions and to examine the behavioral mechanisms underlying learning outcomes.

F1. Trace Data Overview

The tutoring session yielded 3,391 chat messages (881 from students, 2,510 from tutors), with students in the Proactive condition receiving more system support. In total, 1,596 triggers were recorded in the Proactive condition, primarily consisting of idle-state triggers, followed by encouragement and error-state triggers. Across 711 tracked code submissions, 333 (44.18%) resulted in errors, and 20 students concluded their sessions with unresolved issues. All events were precisely timestamped, enabling the computation of process measures such as resolution latency and post-intervention processing latency. Table F1 reports the full descriptive statistics of the trace data.

Table F1. Trace Data Overview for the Tutoring Session

Trace Component	Value
Total chat messages	3,391
Student-initiated messages	881
Tutor messages	2,510
Mean student messages per student	9.79
Mean tutor messages per student	27.89
Mean tutor messages per student, Proactive condition	44.39
Mean tutor messages per student, Reactive condition	10.64
Total proactive triggers	1,596
Idle-state triggers	1,306
Error-state triggers	42
Encouragement triggers	247
Total code submissions	711
Error-producing submissions	333
Mean error-producing submissions per student	4.27
Mean per-student error rate	44.18%
Students ending with unresolved errors	20
Mean final code length (characters)	530.66

***Note*:** Code-submission descriptives are based on students who executed code during the tutoring session.

F2. Help Articulation

This appendix extends the help articulation analysis in Section 6.2 by decomposing learner messages into finer-grained functional categories. While Section 6.2 establishes how tutor initiation shapes the precision of help requests, this appendix examines how *diagnostic specificity* is expressed across different functional categories of learner messages.

Functional Message Categories. Each learner message was classified into one of five functional categories (i.e., What, How, Why, Operational, and Other) to capture the type of help request. Table F2 provides descriptions and representative examples of messages in each category. The "Other" category aggregates messages that do not map to the primary functional categories, including verification checks, constraints, navigation prompts, code excerpts, and affective or meta-comments. These categories provide a decomposition of help articulation, enabling us to examine whether increases in diagnostic specificity arise from shifts in the types of questions learners ask or from changes in how those questions are formulated.

Table F2. Selected Messages in Each Category		
Category	**Description**	**Real Posts**
How	Procedural questions asking about sequence, steps, methods, or approach	"How would I convert rating to integer"
		"How do I find the index of all instances of the minimum rating?"
		"How would I put the ratings in one list and the games in another list?"
		"How do i calculate it"
		"How would I do it if there are 2 with the same rating"
What	Declarative questions identifying components, data types, structures, or behavior.	"What is wrong with my code"
		"What is the difference between a tuple and a list, and how do you define a tuple?"
		"What does 'item.strip' do?"
		"What functions can be used to find the average rating of the numbers in a list?"
		"What should I do next"
Why	Justificatory questions asking for rationale, reasoning, or correctness.	"Why do I have an error"
		"Why is join important, must you use join"
		"Why are we assuming that if it is even it's a game name"
		"Why does it say this for line 11: ValueError: invalid literal for int() with base 10: '85,'"
		"Why is it not printing the lowest rating"
Operational	Requests for concrete code help, syntax clarification, error resolution, or worked examples.	"Can you give me an example of the split thing"
		"Write the code for me"
		"Can you help me create a loop that sorts games by their average rating and give me the lowest rating along with the name"
		"Fix it"
		"Can you explain the program step by step?"
Other: Verification checks	Student seeks confirmation that their current approach or code is correct.	"Like that?", "Is this better?", "am I on the right track", "did I do it right?", "does this look alright?"
Other: Constraints	Student restricts the solution space by specifying methods to avoid	"Don't use map", "without .join", "using enumerate is too complicated for me", "only using loops, lists, and strings"
Other: Navigation	Single-word or minimal-word prompts asking what to do next.	"Next?", "After that?", "And next?", "Display?"
Other: Code/steps pasted for review	Student submits code, pseudocode, error output, or program output without a formed question.	pasting pseudocode steps, error messages, or sample output for AI to evaluate
Other: Status reports	Student describes current program state or a persistent bug.	"Items is not defined", "I am still only getting Risk", "I'm still doing something wrong"
Other: Affective / meta-comments	Off-task remarks directed at the AI.	"Thank u, that makes sense", "Go xxxx yourself [strong negative language directed at CodeTutor]"

Distribution of Message Categories. Across all messages, the overall distribution of categories is as follows: How (32.5%), Other (28.5%), What (19.3%), Operational (16.6%), and Why (3.1%). Table F3 reports the distribution of message categories by phase (T1: conceptual reasoning vs. T2: code implementation), tutor initiation (proactive vs. reactive), and initial CT proficiency. To highlight shifts in help articulation across phases, Table F3 also reports within-group changes (Δ) from T1 to T2.

In the proactive condition, learners with higher Initial CT Proficiency show decreases in What (−10.3%) and How (−12.4%) messages, alongside increases in Operational (+9.4%) and Other (+11.8%) messages. Although these learners used more structured inquiries during the planning phase, the transition to code implementation prompted more debugging-oriented and open-ended interactions. The most notable change

occurred in the proactive/low-proficiency group, where What inquiries nearly tripled from 8.8% to 24.1%, while Other messages decreased by 11%. In contrast, learners with lower Initial CT Proficiency show an increase in What (+15.3%) messages and a decrease in Other (−11.2%) messages. These learners formulated significantly more technical and specific help requests when engaged with executable code. This pattern supports the rescuing mechanism, suggesting that the combination of code implementation and proactive scaffolding assists novices in identifying specific problem areas.

Table F3. Help Articulation Distribution

Group	Phase	N Msgs	What%	How%	Why%	Operational %	Other %
Proactive/High	**T1**	33	27.3	54.5	3	0	15.2
	T2	159	17	42.1	4.4	9.4	27
	Δ		−10.3	−12.4	1.4	9.4	11.8
Proactive/Low	**T1**	34	8.8	38.2	2.9	17.6	32.4
	T2	137	24.1	38	1.5	15.3	21.2
	Δ		15.3	−0.2	−1.4	−2.3	−11.2
Reactive/High	**T1**	30	13.3	26.7	0	23.3	36.7
	T2	185	22.2	28.6	3.2	18.4	27.6
	Δ		8.9	1.9	3.2	−4.9	−9.1
Reactive/Low	**T1**	18	11.1	44.4	0	16.7	27.8
	T2	176	18.8	25.6	5.1	15.9	34.7
	Δ		7.7	−18.8	5.1	−0.8	6.9

In the reactive condition, learners with higher Initial CT Proficiency exhibit modest increases across What (+8.9%), How (+1.9%), and Why (+3.2%) categories, with decreases in Operational (−4.9%) and Other (−9.1%). Learners with lower Initial CT Proficiency show a decrease in How (−18.8%) messages, representing the most significant shift observed, accompanied by increases in What (+7.7%), Why (+5.1%), and Other (+6.9%) categories. These patterns reinforce the interpretation that reactive initiation preserves learner-driven diagnosis. More advanced learners generate increasingly precise and varied forms of inquiry, whereas less experienced learners show weaker shifts toward structured diagnostic articulation. Across all groups, the proportion of Why messages remains low, not exceeding 5.1% in any phase or condition. This suggests that improvements in diagnostic specificity are driven primarily by increases in problem-specific What and Operational queries rather than by deeper Why reasoning. As such, the mechanism operates through improved problem identification and localization rather than through increased explanatory depth.

F3. Error Recovery

This appendix extends the error recovery analysis in Section 6.3 by examining the dynamics of wheeling episodes and the robustness of recovery patterns across alternative definitions. It provides additional detail on recovery trajectories and sensitivity to wheeling thresholds. Figure F1 visualizes survival differences in wheeling recovery across modes and Initial CT Proficiency. The Kaplan-Meier curves show the proportion of students still wheeling over time, allowing comparison of recovery rates between Proactive and Reactive conditions.

Sensitivity Analysis. Table F4 reports sensitivity checks using alternative measures of wheeling onset, ranging from more liberal to more conservative thresholds for identifying sustained error states. The purpose of these checks is to assess whether the recovery results depend on a single event-definition choice. To the extent that the substantive pattern remains stable across these alternative specifications, the results provide greater confidence that the observed differences in resolution latency reflect the underlying tutoring dynamics rather than an artifact of the chosen wheeling threshold.

These supplementary results clarify that proactive tutoring improved recovery efficiency, especially among students with higher Initial CT Proficiency, while longer self-directed wheeling in the reactive condition was associated with sharper subsequent help articulation.

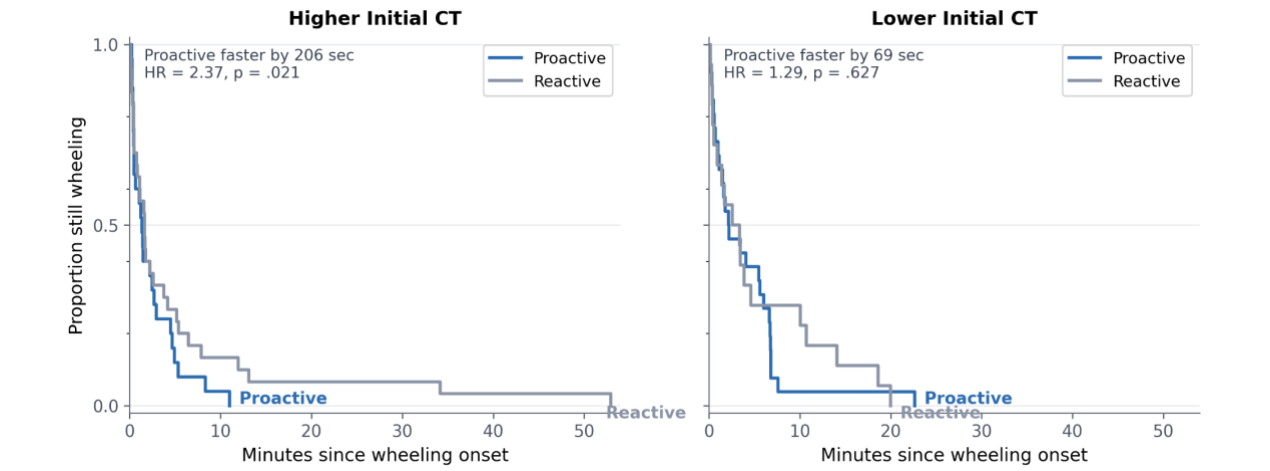


Figure F1. Kaplan–Meier Curves for Wheeling Recovery by Mode × Initial CT Proficiency

Table F4. Sensitivity of Wheeling Results to Alternative Onset Definitions

Onset Definition	Metric	Eps.	N	Higher Initial CT Proficiency			Lower Initial CT Proficiency		
				sec	ratio	*p*	sec	ratio	*p*
Consecutive errors (most liberal)	Seconds to first successful execution	99	42	-186.44	2.30	.18	-86.14	1.36	.40
Consecutive errors + similarity >= .90		86	41	-198.87	2.52	.20	-64.56	1.24	.57
Consecutive errors + similarity >= .95		79	41	-182.40	2.24	.28	-48.71	1.18	.70
Consecutive errors + similarity >= .98		68	39	-220.62	2.56	.29	-10.97	1.03	.93
Consecutive error-indicating chat messages	Seconds between error messages	40	22	68.95	.51	.25	15.92	.90	.81
Error-rate > 25% threshold	Mean recovery steps	57	57	.35	.91	.60	-.18	1.05	.75

F4. Solution Navigation

This appendix extends the solution navigation analysis in Section 6.4 by providing additional detail on how solution path inertia is operationalized through path retention and structural switching, and robustness checks for the measurement of switching behavior. We analyze solution navigation using two complementary measures: structural switching (and its complement, path retention), which captures how much learners shift toward tutor guidance, and processing latency, which captures the temporal effort associated with that shift.

Structural Switching and Path Retention. Figure F2 provides representative code-trace examples of the two response patterns identified in the Solution Navigation analysis. The path-retention examples show cases in which students incorporated a tutor intervention without substantially departing from their prior solution structure. The structural-switching examples show cases in which students depart from their prior solution structure by reorganizing their code toward an alternative solution approach implied by the tutor guidance. These illustrations make the switching logic concrete and clarify how the CodeBERT-based thresholds map onto observable differences in student code trajectories.

Classification Procedures and Threshold Specification. Structural switching is operationalized as a transition in solution strategy, defined by the joint presence of adoption and divergence. Adoption refers to alignment between the learner's code and the tutor-provided solution structure, whereas divergence refers to departure from the learner's prior implementation. We use CodeBERT embeddings to capture the semantic similarity of code representations. Accordingly, structural switching represents reductions in solution path inertia, whereas path retention reflects its persistence. A switching event is identified when post-intervention code exhibits non-trivial adoption of the tutor-provided solution (adopt > .05) and

meaningful divergence from the learner's prior code (pre–post similarity < .85). Cases that do not meet these criteria are classified as path retention.

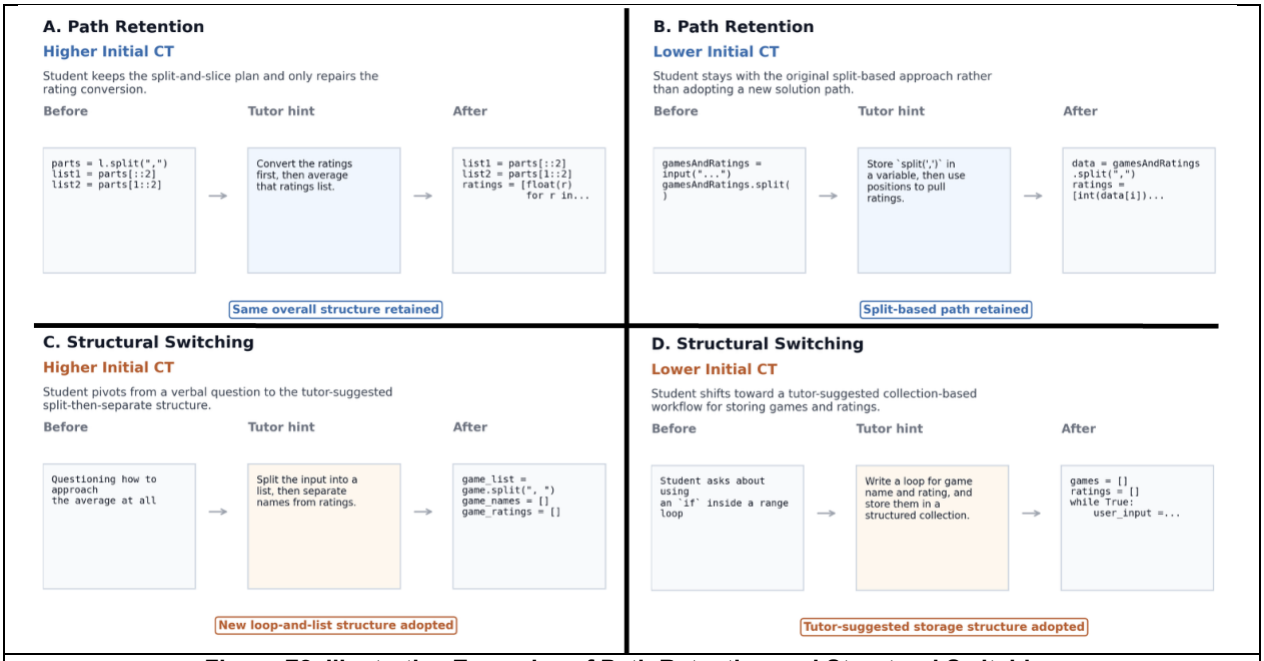


Figure F2. Illustrative Examples of Path Retention and Structural Switching

This conjunctive criterion is necessary for construct validity. Adoption alone cannot distinguish superficial overlap from structural incorporation, and divergence alone may reflect unguided exploration rather than tutor-directed change. The thresholds were calibrated to balance sensitivity and specificity, where the adoption cutoff filters incidental token-level overlap, and the divergence cutoff excludes minor edits that preserve underlying solution logic.

Process Latency. Processing latency measures the time interval between a tutor intervention and the learner's subsequent action, computed from timestamped interaction logs. This measure captures the temporal effort required to evaluate and respond to tutor guidance.

Sensitivity Analysis. Table F5 reports sensitivity of structural switching rates to alternative CodeBERT threshold specifications. These thresholds vary the criteria for identifying adoption of a new solution structure, ranging from more liberal to more conservative definitions.

To the extent that the higher switching rate among novice learners persists across alternative cutoffs, the results provide greater confidence that the observed pattern reflects meaningful differences in solution navigation rather than an artifact of a single threshold specification.

Table F5. Sensitivity of Structural Switching Rates to Alternative CodeBERT Thresholds

Check	Threshold specification	Higher Initial CT		Lower Initial CT		p
		Switching Rate	n	Switching Rate	n	
Original reported rates	adopt > .05; pre–post similarity < .85	1.20%	2	13.50%	47	< .001
Very liberal threshold	adopt > .00; pre–post similarity < .95	17.40%	29	33.70%	117	< .001
Moderate	adopt > .05; pre–post similarity< .95	9.60%	16	15.00%	52	< .1
High adoption threshold	adopt > .20; pre–post similarity < .85	.60%	1	8.40%	29	< .001
Very high threshold	adopt > .25; pre–post similarity < .85	.00%	0	4.60%	16	< .01

F5. Simulation Procedures for Moderated Mediation

To complement the student-level mediation analysis reported in §6.6, this appendix evaluates the statistical sensitivity and directional stability of the moderated mediation results. Specifically, we assess power to detect moderated indirect effects at the observed sample size and examine the extent to which the observed directional patterns would be recovered under repeated sampling. These analyses provide additional context for interpreting the absence of statistically significant indirect effects in the main student-level models.

Power Analysis. We first conducted a sensitivity-based power assessment calibrated to the structure of the moderated mediation models estimated in §6.6. With N = 90, the study has approximately 80% power to detect medium-sized effects in relatively simple tests, such as $r \approx .29$ for a two-sided association or $d \approx .60$ for a balanced two-group contrast. Moderated mediation models are substantially more demanding because they require estimation of indirect effects, interaction terms, and bootstrap-based uncertainty intervals. As a result, detectable effects are larger and required sample sizes are substantially higher than for direct-effect tests. In the present design, the required sample size varies across pathways. Some stronger pathways are detectable with samples in roughly the 200-300 range, whereas weaker or more complex moderated indirect effects may require samples closer to 500 or more. Accordingly, the absence of statistically significant indirect effects in §6.6 should be interpreted in light of these power constraints rather than as evidence against the presence of an underlying mechanism.

Monte Carlo Power Simulation. To assess statistical sensitivity more directly, we conducted Monte Carlo simulations calibrated to the observed student-level mediation structure. Simulated datasets were generated from the fitted mediator and outcome equations, using the observed moderator distribution and residual variation from the estimated models. For each simulated dataset, the moderated mediation model was re-estimated and evaluated using bootstrap confidence intervals for the index of moderated mediation and for conditional indirect effects at low (-1 SD) and high (+1 SD) levels of Initial CT Proficiency. Simulations were repeated across a range of larger sample sizes to assess how power changes with increasing N. The results show that power to detect moderated indirect effects is low at N = 90 but increases meaningfully with larger samples for the strongest pathways, particularly the Task 1 resolution-latency pathway and the Task 2 structural-switching pathway. Figure F3 reports these power curves.

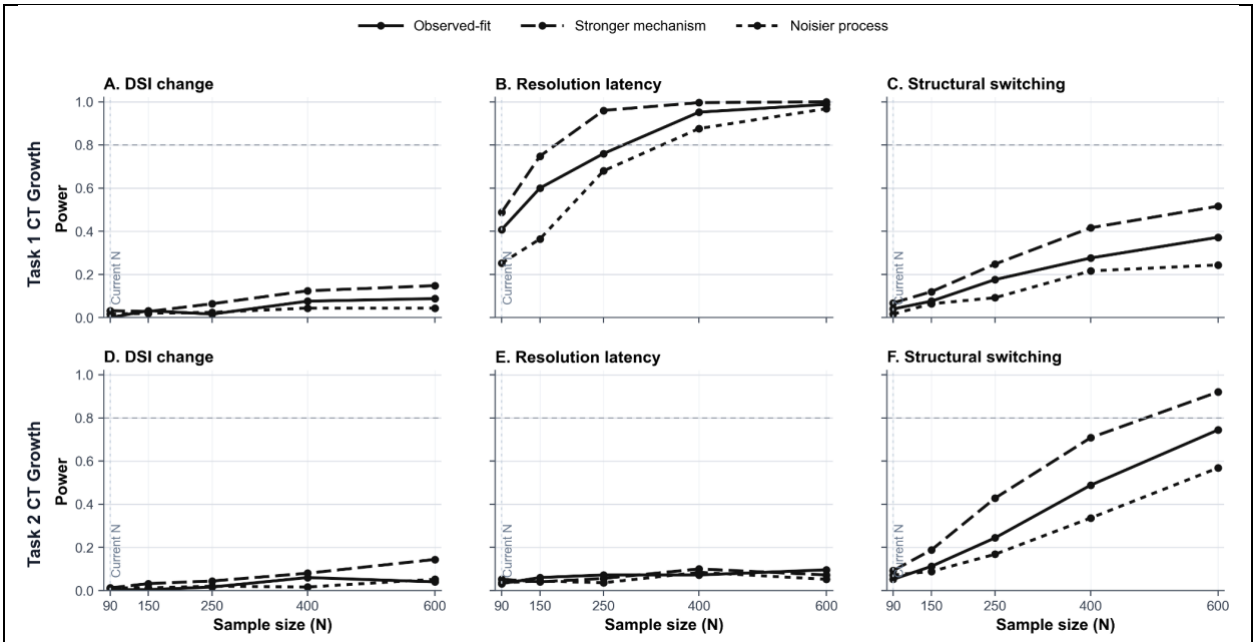


Figure F3. Monte Carlo Power for Moderated Indirect Effects

Assurance Simulation for Directional Stability. To assess whether the observed directional patterns are likely to persist in larger samples, we conducted an assurance simulation that treats the estimated parameters as uncertain rather than fixed. The original dataset was bootstrap-resampled to generate plausible parameter sets, and follow-up datasets were then simulated from those resampled parameter values. For each simulated follow-up study, the moderated mediation model was re-estimated. We recorded whether the estimated effect recovered the same directional sign as the bootstrap-implied effect and whether the corresponding confidence interval excluded zero in that same direction. The results indicate that, for the stronger pathways, directional sign recovery is substantially more likely than conventional CI-based detection. Even when statistical significance remains difficult to achieve at modest follow-up sample sizes, the same directional pattern is often preserved. Figure F4 reports assurance-based replication probabilities.

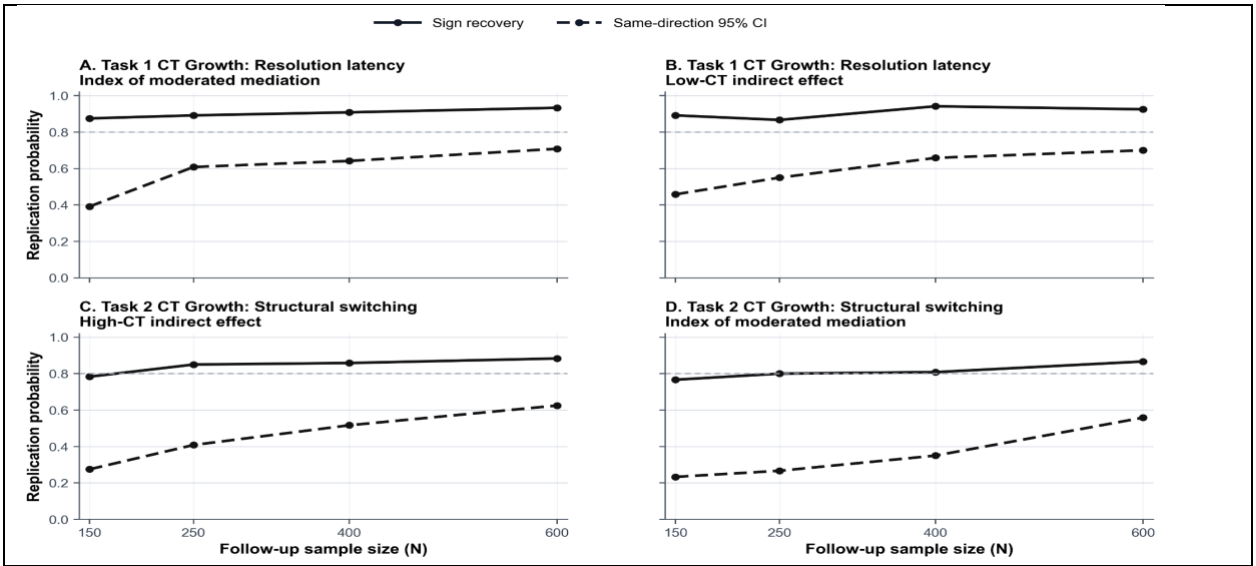


Figure F4. Assurance Simulation for Replicating the Same Directional Effects

These simulations indicate that the student-level mediation results in §6.6 are constrained primarily by limited statistical power, especially for moderated indirect effects. Although the indirect effects are not statistically significant in the current sample, the simulations suggest that the strongest directional patterns would often be preserved in larger follow-up studies, even when formal mediation tests remain demanding. These findings support interpreting the student-level results as suggestive evidence consistent with an underlying process mechanism linking tutor initiation to learning outcomes, while recognizing that the current design is not sufficiently powered to conclusively establish moderated mediation effects.