ClassAid: A Real-time Instructor-AI-Student Orchestration System for Classroom Programming Activities

Gefei Zhang
Zhejiang University of Technology
Hangzhou, China
gefeizhang943@gmail.com

Guodao Sun*
Zhejiang University of Technology
Hangzhou, China
guodao@zjut.edu.cn

Meng Xia†
Texas A&M University
College Station, USA
mengxia@tamu.edu

Ronghua Liang
Zhejiang University of Technology
Hangzhou, China
rhliang@zjut.edu.cn

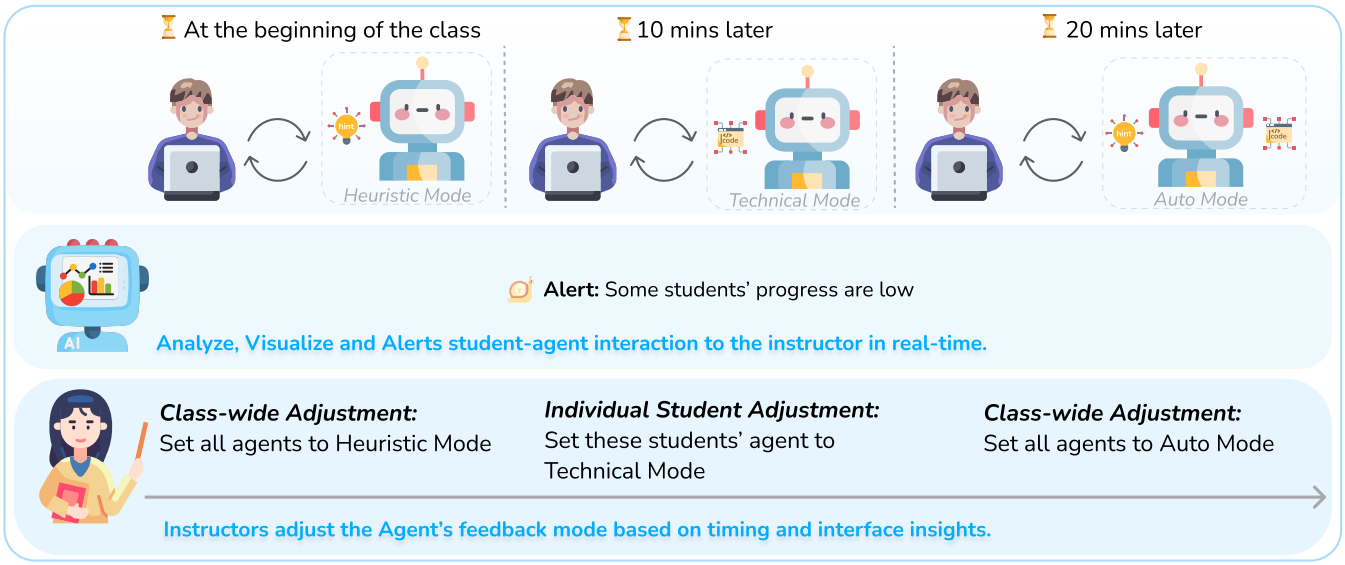

Figure 1: A real-world classroom example of using the *ClassAid* system is shown here. At the beginning of the class, the instructor set all TA agents to Heuristic Mode, which provides high-level hints to encourage independent thinking. After 10 minutes, noticing that some students were progressing slowly, the instructor switched their agents to Technical Mode, which offers code examples. At the 20-minute mark, all agents were changed to Auto Mode, allowing the system to adaptively support students based on their real-time performance.

Abstract

Generative AI is reshaping education, but it also raises concerns about instability and overreliance. In programming classrooms, we aim to leverage its feedback capabilities while reinforcing the educator's role in guiding student–AI interactions. We developed *ClassAid*, a real-time orchestration system that integrates TA Agents to provide personalized support and an AI-driven dashboard that visualizes student–AI interactions, enabling instructors to dynamically adjust TA Agent modes. Instructors can configure the Agent to provide technical feedback (direct coding solutions), heuristic feedback (hint-based guidance), automatic feedback (autonomously selecting technical or heuristic support), or silent operation (no AI support). We evaluated *ClassAid* through three aspects: (1) the TA Agents' performance, (2) feedback from 54 students and one instructor during a classroom deployment, and (3) interviews with eight educators. Results demonstrate that dynamic instructor control over AI supports effective real-time personalized feedback and provides design implications for integrating AI into authentic educational settings.

*Corresponding author.
†Corresponding author.

CHI '26, Barcelona, Spain

ACM ISBN 979-8-4007-2278-3/26/04
https://doi.org/10.1145/3772318.3790824

CCS Concepts

• **Human-centered computing → Visualization systems and tools**; **Visualization systems and tools**; • **Applied computing → Interactive learning environments**.

Keywords

Programming education, AI assistants, class deployment, orchestration system

ACM Reference Format:

Gefei Zhang, Guodao Sun, Meng Xia, and Ronghua Liang. 2026. ClassAid: A Real-time Instructor-AI-Student Orchestration System for Classroom Programming Activities. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (CHI '26), April 13–17, 2026, Barcelona, Spain.* ACM, New York, NY, USA, 27 pages. https://doi.org/10.1145/3772318.3790824

1 Introduction

Programming is increasingly recognized as a foundational literacy of the digital era, swelling beginner enrollments and straining the capacity of large courses to provide timely, individualized feedback [35, 41]. Conventional supports can be insufficient for novices, as limited instructor time, help-seeking hesitancy, and repeated requests from a small subset of students often result in uneven feedback [32, 42, 82, 83]. Generative AI, particularly large language models (LLMs) such as ChatGPT, has opened significant opportunities for programming education by improving information retrieval, serving as capable programming assistants, and reducing instructor workload [50, 52, 74]. LLMs achieve high accuracy on beginner tasks and readily complete small programming exercises [28, 30]. Despite this promise, classroom use remains challenging due to persistent AI instability and student overreliance [27, 42], highlighting the need for more reliable, pedagogically aligned support systems. Systems such as CodeAid leverage LLMs to provide accurate, solution-free feedback for post-class learning, yet they lack mechanisms for real-time instructor monitoring and adaptive adjustment to students' evolving progress [32]. Other systems, such as SPHERE, utilize LLMs to help instructors generate large-scale, high-quality personalized feedback. However, they did not analyze how students interact with LLM-driven agents and remain insufficiently dynamic to adjust responses in real time. Nonetheless, none of these systems is designed to continuously interpret ongoing student–AI interactions or to support instructors in real-time orchestration of the AI's behavior [37], which is essential for preventing overreliance and enabling more targeted, personalized support [36, 37, 40].

Moreover, most LLM-based programming assistants remain passive, responding only to explicit prompts [42, 43]. In contrast, human instructors in classrooms actively circulate, detect when students are stuck or disengaged, and intervene proactively [76]. To mitigate AI passivity and uncertainty while ensuring responsible use, AI systems should incorporate instructor-like reasoning while remaining under instructor control, allowing it to adapt to students' evolving learning states [45]. Accordingly, there is a need for classroom-oriented AI that operates under instructor oversight to provide real-time, personalized feedback, reduce instructors' workload, and foster student engagement [7].

Based on formative and dynamic assessment theories [4, 48], we developed an intelligent TA Agent within the *ClassAid* student interface to deliver personalized and adaptive support. The agent continuously *monitors* students' interactions with AI to *identify* metacognitive levels and potential obstacles, *reviews* prior work and current performance to diagnose issues and assess progress, *considers* alternative feedback responses, *selects* the one most aligned with the student's current needs, and implements targeted *interventions* to help students adjust strategies and strengthen metacognitive abilities. In parallel, *ClassAid* provides an instructor dashboard that offers real-time visibility into student–AI interaction patterns and the TA Agent's response. Instructors can dynamically regulate the Agent's behavior by switching among four feedback modes (*technical, heuristic, automatic, silent*), ensuring pedagogically aligned guidance while preventing student overreliance on AI. We deployed *ClassAid* in a class activity with 54 students, where the instructor actively monitored and adjusted Agent behaviors through the dashboard, demonstrating the system's practicality and effectiveness in authentic teaching scenarios. Follow-up interviews with eight programming educators further validated its instructional value and highlighted its potential for broader application. The contributions of this study are summarized as follows:

- We propose a six-stage intelligent TA Agent framework based on formative and dynamic assessment theories [4, 48], which operationalizes instructors' diagnostic reasoning into dynamic analysis and personalized feedback.
- We develop *ClassAid*, an instructor-AI-student orchestration system that analyzes students' interactions with AI in real time and enables instructors to dynamically adjust the Agent's behavior for personalized support while preventing overreliance.
- Through classroom deployment (n = 54), TA Agents' feedback-quality evaluation, and educator interviews (n = 8), we demonstrate that *ClassAid*'s instructor–AI collaboration provides effective real-time personalized support for classroom programming.

2 Related Work

2.1 LLM-based Conversational Agent in Programming Education

LLMs show promise in programming education by automating content generation [11], improving error explanations and debugging [57], and enabling strategies such as AI-guided learning and code refactoring [41]. But instructors worry that, especially for beginners [31, 53], AI's excessive helpfulness can foster overreliance and weaken critical thinking [19]. For example, novices using tools like GitHub Copilot embedded in the IDE may quickly accept automatic suggestions without understanding the underlying logic, leading to passive engagement [10].

To address this issue, researchers design "guardrails" to ensure academic integrity and promote meaningful learning [42]. For instance, Liu et al. developed CS50.ai, which not only provides fast and accurate AI-generated answers but also incorporates "teaching guardrails" to encourage students to think critically rather than simply providing answers [12]. Similarly, CodeAid designed

various query functions to prevent AI from directly offering solutions [32]. Although such coding assistants avoid giving direct answers [12, 32], existing systems still lack human-like teaching interaction capabilities and exhibit insufficient intelligence. A key limitation is that outputs are tied to fixed prompts and adapt poorly to evolving student needs in real class[45]. Most systems also keep AI reactive, responding only to explicit requests. Recent studies explore proactive code detection and help [8, 43], but their feedback often remains oriented toward task completion and weakly aligned with specific learning objectives [1].

Instructors' support during classroom programming is dynamic and context dependent [64]. It must account for students' cognitive level, learning objectives, progress, and overall class performance, which calls for personalized and adaptive guidance [49]. For example, SPHERE leverages students' process-based evidence to help instructors craft personalized feedback [65], but its instructor-driven, asynchronous workflow requires manual review, which prevents it from delivering real-time or adaptively responsive support to students' evolving needs or their interactions with AI. This paper examines how to design more intelligent AI that can proactively participate in students' work in real-time while remaining dynamically adjustable and supervisable by instructors in classroom programming.

2.2 Classroom Orchestration in Education

2.2.1 Real-Time Classroom Orchestration. Classroom orchestration encompasses individual assignments [21], group tasks, and whole-class interactions [44]. Instructors need to plan [15], monitor, and adjust activities to achieve instructional goals [46]. Real-time orchestration and monitoring are essential for providing timely feedback [23]. For example, FACT monitors learners' behaviors, such as handwriting and typing, to help instructors stay aware of classroom dynamics [69]. Codeopticon displays each learner's actions on a dashboard with real-time collage views of editing and debugging processes, supported by chat for one-to-many tutoring [17]. VizGroup offers group-level anomaly detection through alerts and notifications [66]. Tools such as RIMES [34] and VizProg [81] provide dashboards for real-time observation, enabling instructors to identify key learning behaviors. Building on these systems, SPARK introduces a checkpoint-based progress monitoring framework that dynamically visualizes students' progress across stages [79]. These systems improve instructors' monitoring and analytical efficiency but remain focused primarily on real-time learning analytics, with limited support for in-situ classroom adjustments. *ClassAid* extends this line of work by enabling instructors to not only view learning analytics but also dynamically adjust classroom strategies and feedback in response to emerging instructional needs.

2.2.2 Human–AI Co-Orchestration. With advances in AI, coordination tools now support a human and AI collaborative orchestration model that helps manage complex classrooms [67] [24]. For example, Pair Up in mathematics education automates peer matching and assigns roles such as mentor and problem solver, easing group management [77]. Despite demonstrated benefits, AI-based orchestration tools remain inflexible. They center on lesson-plan adjustments rather than novices' evolving cognitive needs, offering little personalized or adaptive support [25, 71, 72]. Research also seldom examines how students interact with generative AI or how instructors can monitor and manage these interactions in real time [9, 73]. As generative AI becomes increasingly common in classrooms, this gap risks missed timely interventions and raises concerns about the quality, reliability, and safety of interactions between students and AI, including the potential for misinformation, cognitive overload, and overreliance [75] [56]. Instructors, therefore, need real-time insights into individual progress, whole-class dynamics, and how students engage with AI systems [78]. Such visibility enables more purposeful interventions and helps ensure the appropriate, productive, and safe use of AI [33]. To address these challenges, we study students' cognitive processes during classroom programming and introduce a multi-level TA Agent framework. *ClassAid* provides interactive AI guidance for students and a transparent dashboard for instructors to observe and understand interactions between students and AI.

3 Formative Study

We conducted a formative study to investigate challenges in classroom programming and to understand instructors' concerns and expectations about integrating AI into teaching. Seven university programming instructors (T1–T7) were recruited through snowball sampling (4 female; mean age = 35.42, SD = 6.50; teaching and programming experience: mean = 8.14 years, SD = 6.67), each receiving $20 compensation. Additional demographic details are provided in the Appendix. The study was approved by the host university's IRB.

3.1 Procedure

We conducted semi-structured Zoom interviews to understand challenges in classroom programming and to elicit instructors' needs and feedback. The protocol covered experiences with in-class programming, instructional design goals, assessment practices, and common difficulties. We also explored attitudes toward using AI in class and related concerns, posing follow-up questions as needed to obtain details (shown in the Appendix A.2). Each session lasted 40–60 minutes and was documented through typed notes and audio recordings.

3.2 Findings

The following summarizes instructors' perspectives on programming challenges and on using AI tools to support teaching.

3.2.1 Challenges in In-Class Programming Activities. **C1: Limited Feedback Capacity – Challenges in Providing Timely and Personalized Support.** Consistent with prior work, participants reported that limited class time hindered timely and personalized feedback. Peer assessment offers some support but does not meet students' personalized needs [70]. *T1 noted that only students who proactively seek help receive responses, leaving quieter students overlooked. Participants linked low help-seeking to social pressures, such as fear of seeming "not smart." T7 found that anonymous question submission increased willingness to ask but did not allow real-time responses. T5 used dedicated Q&A segments to batch questions, but the lack of continuity limited ongoing support.* Without timely feedback, students may fall into cycles of misunderstanding, and instructors often detect issues only after they escalate.

C2: Instructional Blind Spots – Difficulty Monitoring Student and Class Progress. Consistent with prior work, instructors reported difficulty monitoring progress and task completion in real time, limiting immediate pedagogical adjustments [66, 81]. *T2 noted that large-class settings often create a "disconnect" that prevents instructors from adjusting instruction to students' needs.* Participants also struggled to identify common classwide issues. These challenges highlight the need for real-time support that captures individual and class progress and provides efficient, personalized feedback.

3.2.2 Challenges of Using AI Tools in Real-World Classrooms. **C3: Lack of Trust – Concerns About the Accuracy and Appropriateness of AI-Generated Content.** All participants had prior experience with AI and acknowledged its potential for programming feedback, but they questioned the reliability and pedagogical appropriateness of AI outputs. They cited risks of biased or incorrect content, academic dishonesty, and overreliance [29]. Emphasizing that the process matters more than the answer [19], *T7 allowed AI use but required students to submit conversation logs to monitor motivation and ability.* Participants also worried that AI can weaken higher-order thinking. *T2 noted that some students become accustomed to "mechanical questioning and mechanical receiving," which undermines independent and critical reasoning.*

C4: Limited Intelligence – Inability of AI to Offer Dynamic and Contextualized Feedback. Participants noted that current AI tools lack the feedback flexibility and contextual awareness that human instructors provide. Whereas instructors integrate student background, task progress, and course pacing when adapting support, AI tools often generate fixed responses based on preset prompts and cannot adjust in real time. Instructors also vary their guidance throughout an activity, starting with broad encouragement and shifting to detailed, directive support. T4 commented, *"AI is usually passive, but classroom interactions are active. Instructors intervene proactively; AI still struggles with that."*

C5: Role Conflict – Risk of Undermining Instructor Authority. When we introduced the idea of a TA Agent for real-time feedback, several participants worried it could undermine their classroom authority and marginalize their role. As T5 said, *"If students can get timely and accurate feedback from AI, then what do they still need us for?"* They also noted that AI lacks social pressure, causing some students to favor AI over instructors. T6 remarked, *"After asking the AI, students stop coming to me. They just go straight to the AI."* These shifts reduce instructors' control over pacing and depth and limit their ability to adjust to class needs.

These challenges highlight the need for classroom AI that preserves instructors' authority, supports rather than replaces them, and offers context-aware, adaptive feedback that is targeted and instructionally meaningful.

3.3 Design Goals

Drawing on participant feedback and related research, we propose the following four design goals.

DG1: Provide Real-Time Personalized Feedback to Support Student Learning. Instructors struggle to provide timely, individualized feedback during class due to limited time and large class sizes (C1). *ClassAid* should deliver a TA Agent that offers real-time, context-aware feedback tailored to each student's questions, cognitive level, and learning progress. To address concerns about AI quality and pedagogical appropriateness (C3, C4), the system must ensure feedback is trustworthy, pedagogically sound, and aligned with instructional goals.

DG2: Enable Real-Time Monitoring of Individual and Class-Wide Student-AI Learning Dynamics. Instructors currently lack effective tools to monitor how students engage with AI during problem solving, including their progress, misconceptions, and reliance patterns (C2). *ClassAid* should offer real-time visibility into both individual student-AI interactions and aggregated class-wide learning dynamics, allowing instructors to detect emerging challenges early, adjust instructional strategies in real-time, and deliver targeted, data-informed interventions.

DG3: Empower Instructors with Control and Oversight of AI-Generated Feedback. Instructors express concerns about losing instructional authority and control when AI agents interact directly with students (C5). *ClassAid* should preserve instructor leadership by providing flexible mechanisms for supervising and adjusting AI behavior in line with pedagogical needs and classroom context. This ensures that AI augments rather than replaces, the instructor's role, maintaining accountability and alignment with learning objectives.

DG4: Facilitate Student Engagement and Feedback to Build Trust. Students' acceptance and trust in AI-generated feedback directly impact learning outcomes. To address concerns about AI intelligence and appropriateness (C3, C4) while strengthening the instructor-student feedback loop (C5), *ClassAid* should enable students to actively evaluate and respond to AI feedback. This bidirectional feedback mechanism enhances student agency, provides instructors with signals to assess AI reliability, and fosters a harmonious learning environment.

4 ClassAid Design and Implementation

We designed *ClassAid*, a real-time orchestration system that integrates a student interface (Fig. 2) and an instructor dashboard (Fig. 4) to support programming instruction in live classroom settings.

4.1 Student Interface Design

Once the programming activity begins, students access an interactive interface (Fig. 2). The top of the interface displays the current TA Agent feedback mode (Fig. 2-a1), helping students understand how the agent will respond. Before writing any code, students can view the task description in the Task Panel (Fig. 2-a2), open datasets through the "Data" button (Fig. 2-a3), and preview the expected output via the "Final Visualization" button (Fig. 2-a4). The interface also provides partial starter specifications that allow students to edit an existing JSON structure (Fig. 2-c1), helping them begin the task more effectively. Together, these features help students clearly understand the task objectives and available resources.

During the problem-solving process, if students encounter difficulties, they can ask questions to the TA Agent through the Chat Panel (Fig. 2-B, b3). Drawing on formative and dynamic assessment theories [4, 48], the TA Agent incorporates instructors' adaptive teaching strategies. By analyzing each learner's cognitive level,

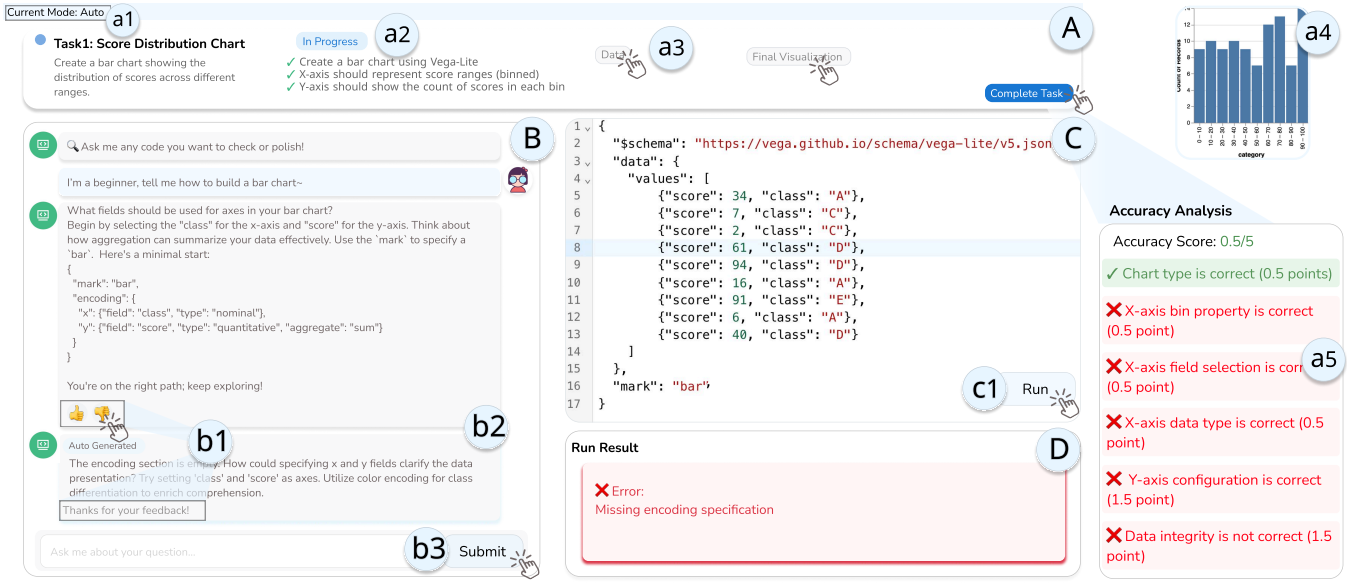

Figure 2: Student interface during in-class programming activities. (A) The task panel shows the current feedback mode (a1), task description (a2), data (a3), and expected output (a4). (B) The chat panel supports student questions and TA Agent responses, with options to rate messages and receive proactive feedback (b1). (C) The code panel allows students to write and run code. (D) The output panel displays execution results and error messages.

error type, and learning progression, it dynamically adjusts feedback to close feedback gaps, support autonomous learning, and promote higher-order thinking while preserving independent reasoning **(DG1)**. The Agent provides four response modes: Heuristic Mode encourages reflection through open-ended prompts; Technical Mode provides concrete code solutions guidance; Auto Mode balances both approaches based on real-time context; and Silent Mode intentionally withholds responses to maintain student autonomy (details in Section 4.2.4). The TA Agent responds according to the active mode. *ClassAid* also allows students to evaluate TA Agent feedback, strengthening the student–instructor feedback loop on AI-generated content. Students can "like" or "dislike" the response (Fig. 2-b1), helping instructors monitor the TA Agent's performance **(DG4)**. Providing feedback is optional, if a student submits a rating, a confirmation message ("Thanks for your feedback!") appears.

Code editing occurs in the Code Panel (Fig. 2-C), where students write their solutions and execute them using the "Run" button (Fig. 2-c1). The execution results are shown in the Output Panel (Fig. 2-D). If an error occurs, the system returns a specific message; if the code runs successfully, the result appears. Unlike typical console reports that provide only generic feedback, our system classifies errors into five categories (schema, data, mark, encoding, and JSON syntax) and offers targeted recommendations. For example, the code in Fig. 2-B would yield only "`Error: Invalid Vega-Lite specification`" in a standard console, which provides limited guidance for beginners. In our interface (Fig. 2-D), the system instead reports "`Error: Missing encoding specification`," explicitly identifying the missing encoding declaration and helping students fix the issue more efficiently.

Except in Silent mode, the TA Agent can also generate proactive feedback. For example, if a student submits incorrect code multiple times within a short period, the system automatically provides suggestions (Fig. 2-b2). This proactive feedback aligns with the active mode: in Heuristic mode it is heuristic, in Technical mode it is technical, and in Auto mode the system selects the appropriate type based on context. These messages are labeled as "Auto Generated" and visually distinguished with a blue background, helping students recognize that the feedback is system-initiated.

Instructors can adjust the TA Agent's feedback mode in real time, and the mode indicator on the student interface (Fig. 2-a1) updates immediately. After finishing a task, students click "Complete Task" (Fig. 2-a5) to move to the next phase. The system then archives the current conversation and code, generates a task summary with core concepts and scoring, and refreshes the interface to create a clean workspace for the next task.

4.2 TA Agent Framework

To build an intelligent TA Agent capable of providing students with real-time, personalized feedback (DG1, DG2), we designed and implemented a six-stage framework grounded in formative and dynamic assessment theories [4, 48]. The framework was developed with guidance from an educational technology expert with over a decade of experience in learning analytics and is powered by LLMs to emulate the diagnostic reasoning loop of human instructors in classroom settings (shown in Table 1). The expert also contributed to defining and validating the weighting of key indicators to ensure their pedagogical soundness. The full prompt templates are provided in the Appendix.

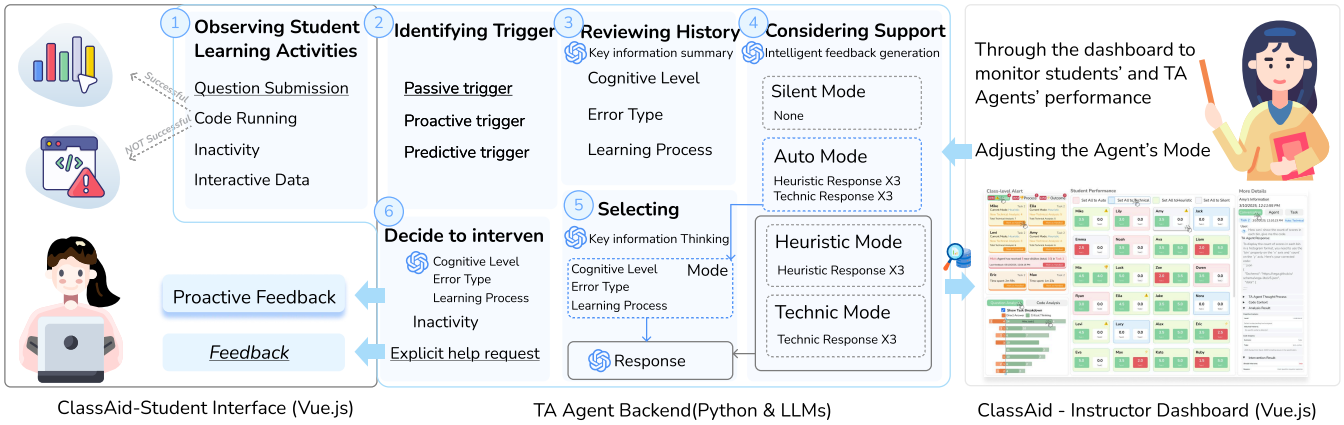

Figure 3: Overview of the TA Agent's six-stage orchestration pipeline for student learning support within the *ClassAid* system. Student activity data, such as question submissions, code execution, and interaction traces, are first collected through the *ClassAid* Student Interface and passed to the TA Agent backend. The agent then enters a six-stage pipeline that observes, analyzes, and responds to student learning behaviors. Meanwhile, instructors monitor student progress and the TA Agent's performance through the *ClassAid* Instructor Dashboard and can adjust feedback modes in real time to align with pedagogical goals.

4.2.1 Stage 1: Observing Student Learning Activities. Once a student begins a task, the TA Agent enters an observation stage and continuously tracks behaviors such as editing inactivity, question submissions, code modifications, and execution events to capture real-time learning dynamics. An inactivity timer with a 240-second threshold flags periods without keyboard, mouse, or click interactions as inactivity. This value was empirically chosen to reduce false alarms when students pause to read or consult materials, helping identify learning bottlenecks or motivational issues. When a question is submitted, the TA Agent classifies it using Bloom's taxonomy (Remember, Understand, Apply, Analyze, Evaluate, Create) [13] and analyzes any accompanying code for syntax or semantic errors, labeling specific error types to clarify student difficulties. Code changes are recorded incrementally. Each time the student clicks "Run," the TA Agent performs a multi-step check that verifies JSON format and required fields, conducts runtime analysis, and returns either error details or the visual output. This integrated monitoring enables precise assessment and personalized feedback.

4.2.2 Stage 2: Identifying Learning Obstacles. Building on Stage 1 behavioral monitoring, the TA Agent in Stage 2 identifies learning obstacles through three types of dynamically triggered interventions. *Passive triggers* respond to explicit student actions, such as question submissions or code execution failures, and are treated as high-priority help requests. *Proactive triggers* arise from the Agent's own judgment. For example, prolonged inactivity or extended editing without queries require autonomous intervention. *Predictive triggers* rely on historical patterns, such as repeated errors or shifts in cognitive level (e.g., a two-level shift across five interactions), signaling potential comprehension gaps. When multiple triggers occur, the TA Agent prioritizes them in the order of passive, proactive, and predictive, ensuring prompt responses to direct help-seeking while still addressing subtler issues. To minimize unnecessary interruptions, we added a time window and cooling mechanism based on inactivity detection. Within any five-minute period, no more than two pause-related triggers are permitted, and non-passive triggers are subject to a two-minute cooling period. If triggered again within this interval, the system returns an empty trigger list and halts further analysis, reducing redundant interventions.

4.2.3 Stage 3: Reviewing and Assessing Student History. Once activated by a trigger, the TA Agent enters the review stage and analyzes the student's historical interactions to support targeted feedback. It automatically retrieves and organizes behavioral data and Student-AI interaction logs, extracting and summarizing key information across multiple dimensions. The Agent monitors the student's cognitive level using Bloom's taxonomy to detect stagnation or regression [13]. It also compiles statistics on error types, frequencies, and distributions to identify recurring patterns and infer potential conceptual misunderstandings. To evaluate task performance, the Agent considers both progress and code execution success to estimate the student's learning stage and mastery level. From a conceptual perspective, it distinguishes between mastered and problematic concepts, highlighting areas that require further study.

4.2.4 Stage 4: Considering Appropriate Forms of Learning Support. Based on our formative study and related educational theories, we found that instructors consider students' cognitive level, error type, and learning progress when deciding how to provide support. In practice, we identified three recurring support patterns. At the beginning of the class activity, instructors often used open-ended and encouraging prompts that guided students to construct understanding independently, consistent with constructivist teaching and aligns with heuristic feedback [22]. As the activity progressed and students encountered concrete syntax or logic problems, instructors shifted to more detailed and directive guidance to help them resolve problems efficiently, corresponding to technical feedback [6]. When

Table 1: Alignment between the Six-Stage TA Agent Framework and Formative / Dynamic Assessment Theories.

Stage	Theoretical Alignment	Implementation
Observing Student Learning Activities	Formative Assessment – Evidence Collection	• Monitor editing, running, questioning, and inactivity • Classify questions with Bloom's taxonomy • Analyze syntax/semantic errors • Inactivity timer flags stagnation
Identifying Learning Obstacles	Dynamic Assessment – Exploring ZPD Boundaries	• Passive triggers: student requests • Proactive triggers: inactivity or extended editing • Predictive triggers: historical patterns, repeated errors • Simulates teacher probing of ZPD
Reviewing and Assessing Student History	Formative Assessment – Accumulating Evidence Dynamic Assessment – Simulating ZPD	• Aggregate logs of past interactions • Track Bloom's taxonomy level shifts (stagnation/regression) • Compile error statistics and distributions • Distinguish mastered vs. problematic concepts • Model ZPD dynamically using historical patterns
Considering Appropriate Forms of Learning Support	Formative Assessment – Feedback and Adjustment	• Generate feedback based on cognitive level, error type, and knowledge gaps • Heuristic feedback: open-ended question, prompts, optional code, supportive tone • Technical feedback: brief explanation, code fix (3–5 lines), respectful tone • Four instructor modes: Auto, Technical, Heuristic, Silent
Selecting Adaptive Feedback Modes	Formative Assessment – Quantifying Diagnostic Judgments	• Auto mode: applies cognitive psychology and instructional strategy framework • Weighted scheme: cognitive level (50%), error type (20%), learning history (30%) • Score candidate feedback: relevance (40%), complexity (20%), consistency (20%), clarity (15%), urgency (5%) • Select the highest-quality response
Intervening to Support Learning Progress	Formative Assessment – Feedback Implementation Dynamic Assessment – Scaffolding and Extending Potential	• If inactivity/help request: immediate intervention • Otherwise compute intervention score (error severity 40%, cognitive level 30%, history 30%) • Score > 0.5 → proactive intervention • Score ≤ 0.5 → support autonomous exploration • Scaffolding extends learning potential

instructors noticed signs of overreliance on AI support, they deliberately delayed or temporarily withheld direct answers to preserve space for autonomous exploration and productive struggle [54, 62]. Building on these findings and on formative and dynamic assessment theories [4, 48], we modeled instructional support as four feedback modes that integrate both feedback type and triggering mechanism: Auto, Heuristic, Technical, and Silent feedback (details in Table 2).

In this stage, the TA Agent generates targeted feedback based on the student's cognitive level, error types, and knowledge gaps. To ensure pedagogically grounded and context-sensitive responses, we developed a structured prompt-based framework that considers two key dimensions: *feedback type* (heuristic vs. technical) and *triggering mechanism* (proactive vs. user-triggered). Examples are shown in Table 3.

(1) **Context-setting Prompt:** Defines the TA's instructional persona and pedagogical goals. For example:
> You are a helpful and encouraging teaching assistant for a Vega-Lite data visualization course. Depending on the situation, you may proactively highlight issues or respond directly to students' questions.

Table 2: TA agent support modes derived from formative classroom observations, aligned with theoretical foundations and design implementations.

Mode Type	Theoretical Alignment	Formative Findings (C4)	Design Implementation
Heuristic Mode	Socratic Method [3]; constructivist Learning theory [22].	At the beginning of the task, instructors often use open ended questions and *Socratic dialogue* to stimulate students' thinking while deliberately avoiding giving direct answers.	The agent uses open ended prompts to trigger reflection, offering directions for thinking instead of direct solutions and guiding students to reason and revise on their own.
Technical Mode	Direct Instruction [6].	When students encounter specific technical obstacles such as syntax errors or logic errors, instructors shift to precise and directive support to ensure that the task can be completed.	The agent provides focused and task specific guidance, including procedural steps, code examples and error corrections, which helps students debug quickly and move the task forward.
Auto Mode	Formative Assessment Theory [4]; Dynamic Assessment [48].	Instructors continuously assess students' states and flexibly adjust support strategies based on real time context such as learning stage, type of difficulty and current performance, switching between different levels and types of help.	The agent intelligently selects between heuristic and technical modes based on context, taking into account learning stage, problem type and students' past performance, in order to dynamically adjust the form and intensity of support.
Silent Mode	Fading of Scaffolding [54]; Metacognition and Self regulated Learning Theory [62].	Classroom observations indicate that frequent immediate help can foster AI dependence and reduce students' independent attempts, so instructors sometimes delay or withhold responses to prompt autonomous exploration.	For a period of time, the agent intentionally does not respond. This encourages students to search, debug and reflect on their own first, avoiding over reliance on AI.

(2) **Heuristic Feedback Prompts:** Encourage critical thinking and reflection.
   - *Proactive:* Short observations and a focused guiding question (under 50 words).
   - *User-triggered:* One open-ended question, 2–3 concise thinking prompts, an optional code snippet, and a supportive tone (under 100 words).

(3) **Technical Feedback Prompts:** Provide immediate, task-specific support.
   - *Proactive:* A brief explanation and one direct code suggestion (3–5 lines).
   - *User-triggered:* A detailed explanation of the issue, a complete code correction, and reasoning for why it works.

(4) **Response Constraints:** Each feedback instance produces three concise responses, written in clear, respectful, and stylistically consistent language.

(5) **Feedback Mode Configuration:** The generation strategy is governed by four instructor-controlled modes: *Auto Mode:* Both heuristic and technical responses (6) are generated; *Technical Mode:* Only technical responses (3) are generated; *Heuristic Mode:* Only heuristic responses (3) are generated; *Silent Mode:* No feedback is provided.

4.2.5 Stage 5: Selecting Adaptive Feedback Modes. In Stage 5, the TA Agent selects the most appropriate feedback from the candidate responses based on the student's current learning state. In Auto Mode, it assumes the role of a senior educational expert and uses a decision framework grounded in cognitive psychology [61] and instructional strategy [51] to determine whether heuristic or technical feedback is more suitable. We operationalize this decision rule using a heuristic weighting scheme derived from our formative study and discussions with course instructors. The scheme considers three factors: current cognitive level, error types, and learning history, with default weights of 50%, 20%, and 30%, respectively, which can be adjusted for specific course contexts. For example, if the student is operating at the Apply level or above, shows mainly design or logic errors, and is making steady progress, the Agent favors heuristic feedbac. In contrast, lower cognitive levels, frequent syntax errors, and fluctuating performance lead to technical feedback.

After determining the mode, the TA Agent evaluates the three candidate responses along five dimensions. These dimensions draw

Table 3: Examples of Technical and Heuristic Responses under Proactive vs. Normal Templates.

Response Type	Proactive Template (auto-triggered)	Normal Template (user-triggered)

Technical	I noticed your chart does not have a Y encoding, which prevents Vega-Lite from drawing the bars. You can try adding an aggregate count on the Y-axis:
	{
	"y": {"aggregate": "count", "type": "quantitative"}
	}
	This small change will allow the bars to show the number of records in your dataset.
		The issue happens because Vega-Lite does not know what to plot on the Y-axis. To make the bar chart display properly, you should specify an aggregate count for the Y encoding. For example:
		{
		"x": {"field": "category", "type": "nominal"},
		"y": {"aggregate": "count", "type": "quantitative"}
		}
		Here, "x" shows categories, and "y" counts how many records fall into each. This way, the height of each bar corresponds to the frequency of that category.

Heuristic	I noticed the Y-axis is not defined. What would happen if you tried adding a count aggregation for Y? Could that make the bars appear as expected?	What do you expect the Y-axis to represent in your bar chart? Should it show raw counts, averages, or something else? If you want counts, you might consider using an aggregate function. Which option best matches the story you want your chart to tell? You are making good progress.

from prior work on effective feedback and instructional design: relevance and clarity reflect established principles of high-quality formative feedback [20]; complexity relates to cognitive load [63]; consistency with prior behavior supports self-regulated learning [68]; and urgency highlights the importance of timely intervention [60]. The default weighting configuration (40%, 20%, 20%, 15%, and 5%) is informed by our formative study and can be adapted to instructional priorities.

4.2.6 Stage 6: Intervening to Support Learning Progress. At this stage, the TA Agent finalizes the selection of feedback mode and content and then enters the decision implementation phase. During this phase, the system must determine whether to intervene in the student's learning process to provide timely support and promote metacognitive development. The Agent first checks for special conditions, such as signs of inactivity or explicit help requests. If any of these conditions are detected, the system intervenes immediately to address learning stagnation or respond to the student's needs.

If none of these signals are present, the Agent activates a motivation-based intervention mechanism. This mechanism conducts a composite assessment based on three factors: cognitive level, error type, and learning history. These factors mirror those used to select the feedback mode, although with different weightings. Instructor interviews show that cognitive level carries the greatest weight when determining the feedback mode, while error type and learning history become more influential during assessment because reveal persistent misconceptions and recurring difficulties.

In contrast, for intervention decisions, instructors emphasized that error type is especially important. If a student is making only simple or low-level mistakes, immediate intervention is neither necessary nor desirable, as allowing more time for independent exploration may better support productive struggle and self-discovery. The weights assigned to the three factors are informed by empirical observations from our formative study and can be adjusted to fit specific course needs. The default weighting configuration is 40%, 30% and 30%. Each factor is normalized to a 0 to 1 range, and a weighted composite score is computed. When the score exceeds the intervention threshold (default = 0.5, configurable), the Agent initiates a proactive intervention; otherwise, it refrains from intervening to preserve autonomous exploration.

4.3 Instructor Dashboard Design

Real-Time Student Overview via Dynamic Performance Cards (DG2). When a student joins the interactive dashboard shown in Fig. 4, a corresponding student card (Fig. 4-C) is generated in the instructor's Student Performance panel. Each card displays the student's name and their current task state. If the student has not completed any tasks, the system shows "—" and the task background remains white. Once a task is completed, a score out of five appears, scores below three are shown in red and scores from three to five in green, providing a intuitive indication of student performance. The card background color also reflects the TA Agent's current feedback mode: purple for Auto Mode, blue for Technical Mode, yellow for Heuristic Mode, and gray for Silent Mode. This design helps instructors quickly understand both student progress and the support they are receiving. Instructors can also manage AI behavior at the class level (Fig. 4-c3). By clicking the "Set Mode to All Students" button, they can apply the selected mode to the entire class in one action, improving operational efficiency and enabling timely adjustments. As shown in Fig. 1, the instructor sets the class-wide AI mode to Heuristic Mode at the beginning of the activity to encourage independent thinking. About ten minutes later, the instructor notices unusual behavior in several students and adjusts their modes individually. In the second half of the class, the instructor switches

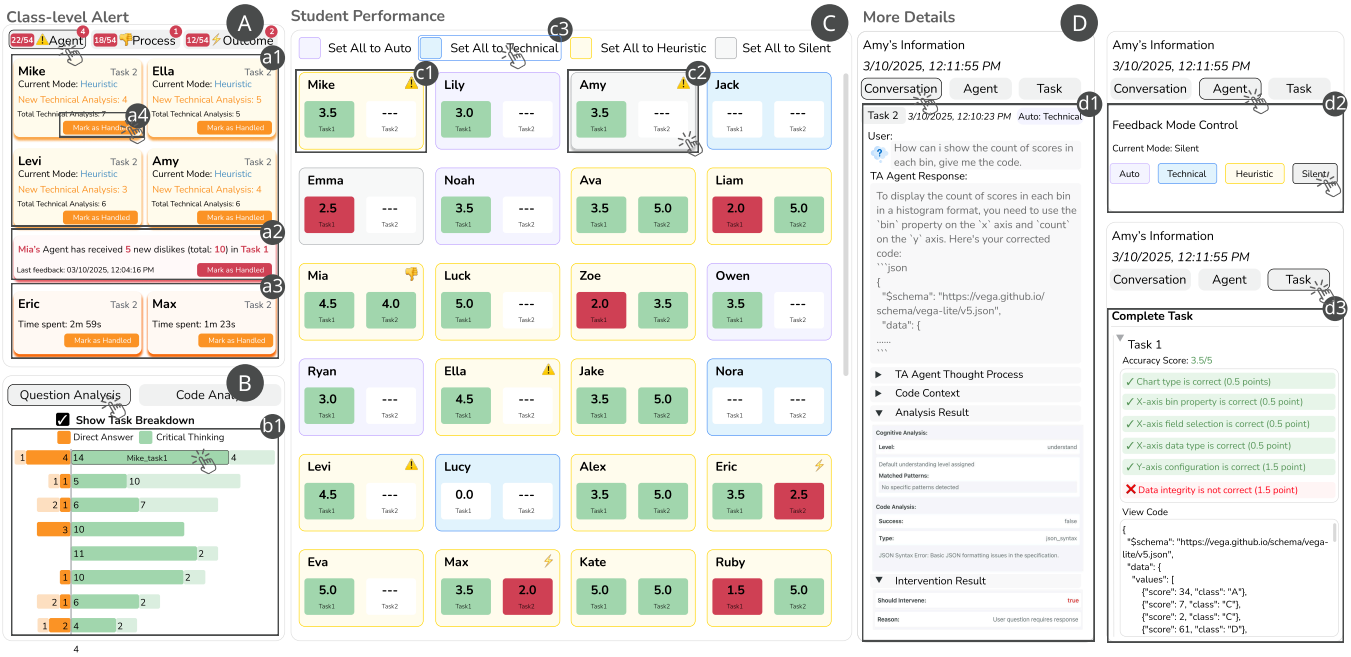

Figure 4: Instructor dashboard for real-time classroom orchestration. (A) Class-Level Alerts highlight potential learning risks through Agent, Process, and Outcome alerts. (B) Class-Level Analysis aggregates question (b1) and code (b2) issues to reveal class-wide bottlenecks. (C) Student Performance Cards display each student's task score and current feedback mode, with global controls for mode switching (c3). (D) More Details Panel provides drill-down views of individual students, including agent interactions (d1), mode control (d2), and task-level analysis (d3). Together, these components enable timely intervention and data-informed teaching decisions.

the entire class to Auto Mode so the system can deliver adaptive support based on real-time performance.

Live Alerts to Surface Critical Teaching Moments (DG3, DG4). As the activity progresses, the system continuously updates two panels, Class-Level Alerts (Fig. 4-A) and Class-Level Analysis (Fig. 4-B). These panels help instructors identify irregular student behaviors and potential anomalies in TA Agent feedback. The alert mechanism supports timely instructional intervention through three types of alerts. Agent Alert (Fig. 4-a1) appears when the TA Agent in Auto Mode produces technical feedback three consecutive times, suggesting potential overreliance on direct answers. Process Alert (Fig. 4-a2) notifies instructors when a student gives three "dislikes" on feedback within a single task, signaling potential mismatches between the AI's support and the student's needs; Outcome Alert (Fig. 4-a3) is shown when a student completes a task in under three minutes, raising concerns about shallow engagement. This threshold was determined in consultation with instructors during the formative study and can be adjusted to meet specific instructional needs. All alert information is reflected on the corresponding student card (Fig. 4-c1), enabling instructors to detect issues quickly. Instructors can toggle across alert tabs to view class-wide summaries. Each tab shows how many students have ever triggered that alert type (e.g., 22 out of 54 in Agent Alert), and a red badge indicates the number of unresolved alerts requiring attention. Once an alert is addressed, instructors can click "Mark as Handled" (Fig. 4-a4) to remove it from view, ensuring only unresolved alerts remain visible.

Class-Level Analysis to Identify Group-Wide Bottlenecks (DG2). To help instructors identify common issues across the class, the Class-Level Analysis panel provides two subviews: Question Analysis and Code Analysis (Fig. 4-B). To monitor students' use of AI and detect abnormal behaviors, we designed a pyramid bar chart that visualizes question types. As shown in Fig. 4-b1, this view uses LLM-based analysis to show whether students are primarily engaging in critical thinking or requesting direct answers. Each bar represents a student: the orange section on the left indicates the number of answer-seeking questions, and the green section on the right represents critical thinking questions. This classification is derived from content analysis of students' submissions. When instructors hover over a bar, they can see the student's name and total interactions. A "Show Task Breakdown" toggle reveals the distribution of question types across tasks. The chart updates dynamically, and students are sorted by total question count, with the most active students listed at the top. In the Code Analysis view, student code errors are categorized and displayed in a bar chart (Fig. 5-b2). Instructors can click an error category to view the list of affected students, sorted by error frequency so that those with the highest counts appear first and can be prioritized for intervention.

Drill-Down View for Diagnosing Individual Student Needs (DG3). When an instructor identifies unusual behavior or performance

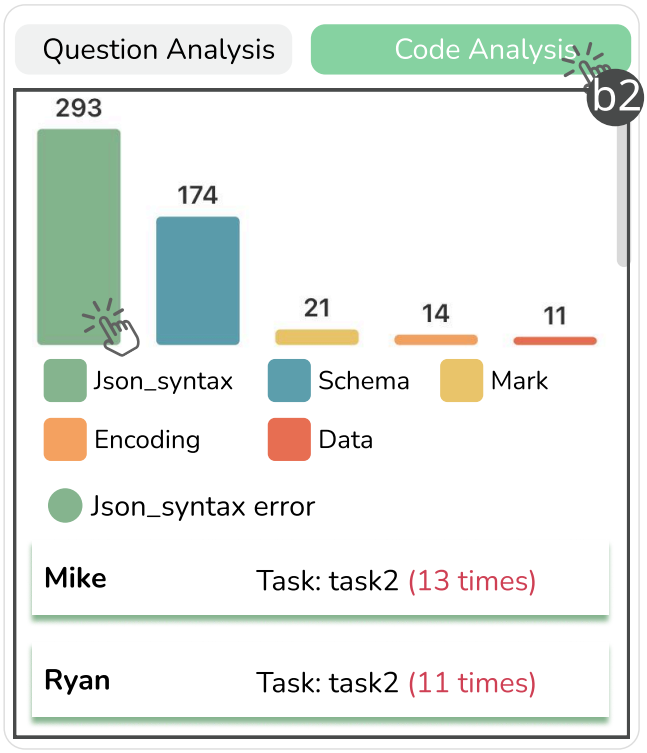

Figure 5: The Code Analysis view shows how often different code problems appear in student submissions. Instructors can click on each bar to see which students had that issue and and how many times it occurred.

issues through alerts or analysis panels, they can click the corresponding student card to open the More Details panel (Fig. 4-D), which provides an in-depth view of the student's learning trajectory and interaction history. This panel includes three subviews. The Conversation Information section logs the full history of the student's interactions with the TA Agent, including the student's questions, the Agent's responses, system-generated cognitive and code analyses, and the feedback mode or intervention applied. This detailed record helps instructors understand the reasoning behind the TA Agent's feedback and supports subsequent instructional decisions (Fig. 4-d1). The Agent Information section allows instructors to view the student's current feedback mode and adjust the TA Agent's support level based on learning progress (Fig. 4-d2) or class pacing (Fig. 4-c3). For example, if class time is nearly over and the student has not made progress, the instructor may switch to Technical Mode to provide more direct code guidance (Fig. 4-c3). Finally, the Task Information panel presents detailed scores and final code submissions for each task. With this comprehensive information, instructors can trace the student's coding process when performance is low, identify learning obstacles and skill gaps, and offer targeted guidance and intervention to the student (Fig. 4-d3).

4.4 ClassAid Implementation

ClassAid is a real-time programming support platform built on a client–server architecture, with a Vue.js frontend and a Python Flask backend. It supports core features including user authentication, interaction logging, and feedback collection. Client–server communication uses RESTful APIs, and Firebase manages user verification and data persistence. The system integrates OpenAI's GPT-4o to perform intelligent code analysis and generate feedback. For further details on the model selection process, refer to Appendix I. As shown in Fig. 3, student activities such as question submissions, code executions, and interface interactions are captured and processed by the TA Agent through a six-stage orchestration pipeline, enabling fine-grained, personalized support. The instructor dashboard provides real-time visibility into student progress and Agent activity. Instructors can adjust feedback modes (Silent, Auto, Heuristic, or Technical) individually or class-wide. These configurations are written to Firebase, which the TA Agent references in real time to align its responses with instructional intent. To support adaptive feedback, *ClassAid* maintains detailed contextual data for each student, including cognitive state changes, code quality metrics, error patterns, question types, learning history, and prior feedback. At each decision point, the system logs agent reasoning, student behavior, triggers, selected feedback, and intervention outcomes. All data is stored and visualized in real-time through the instructor dashboard, and rule-based evaluations prioritize critical events to ensure timely and targeted interventions. This human-AI collaborative design provides instructors with strategic control and immediate insight into student learning, enabling alignment between pedagogical goals and AI-driven support. *ClassAid* thus enhances orchestration capacity in hybrid programming classrooms.

5 Evaluation Design

To evaluate the effectiveness of *ClassAid* in classroom programming and understand student experiences, we employed a mixed-method approach. First, a quantitative study assessed the performance of the LLM-based TA Agent. Second, a classroom study with 54 students, one instructor, and two TAs examined system's usefulness and usability. Finally, semi-structured interviews with eight experienced educators provided insights into its instructional value.

Our study is guided by the following research questions:

Student Interface: How do students perceive the effectiveness of the ClassAid student interface and TA Agent feedback during in-class programming activities?

Instructor Dashboard: How do instructors perceive the accuracy and pedagogical utility of ClassAid's instructor dashboard in real-time classroom monitoring?

5.1 User Study

We conducted a user study in a graduate-level data visualization course at a research university to evaluate *ClassAid*'s usefulness and usability. During a 75-minute session, students completed two Vega-Lite visualization tasks using its declarative grammar for interactive graphics [58].

5.1.1 Participants. Fifty-four computer science graduate students participated (35 males, 19 females; M = 25.95 years, SD = 1.31). The course was taught by a female assistant professor with support from two TAs. Most students had experience with AI tools but were unfamiliar with Vega-Lite. Regarding AI-assisted programming, 49% reported frequent use, 40.8% occasional use, 8.2% limited use, and 6.1% no use. In contrast, 49% had never heard of Vega-Lite, 38.8% had heard of it but never used it, and 12.2% had brief exposure.

5.1.2 Study Design and Procedure. We did not include a baseline system due to methodological and practical constraints. The 75-minute classroom session could not accommodate additional conditions, and repeating tasks risked fatigue and learning effects. A follow-up study with new participants would introduce variability in background and proficiency, making comparisons unreliable.

Before the in-class activity, the instructor delivered a twenty-minute tutorial that introduced Vega-Lite's key concepts and worked examples based on its official documentation, ensuring that students entered the session with essential prior knowledge. The tutorial slides are included in the Supplementary Material. One of the authors demonstrated *ClassAid* and introduced the two tasks (Appendix B.1). Students were given 50 minutes to complete both tasks using the *ClassAid* student interface, with other AI tools prohibited. They completed Task 1 before Task 2 and were informed that their performance would not affect their final grades to encourage natural engagement. Throughout the activity, the instructor and TAs monitored progress and adjusted AI response modes through the instructor dashboard. Students completed a 7-point Likert questionnaire afterward, followed by interviews with 10 students (S1-S10) and post-study interviews with the instructor and TAs.

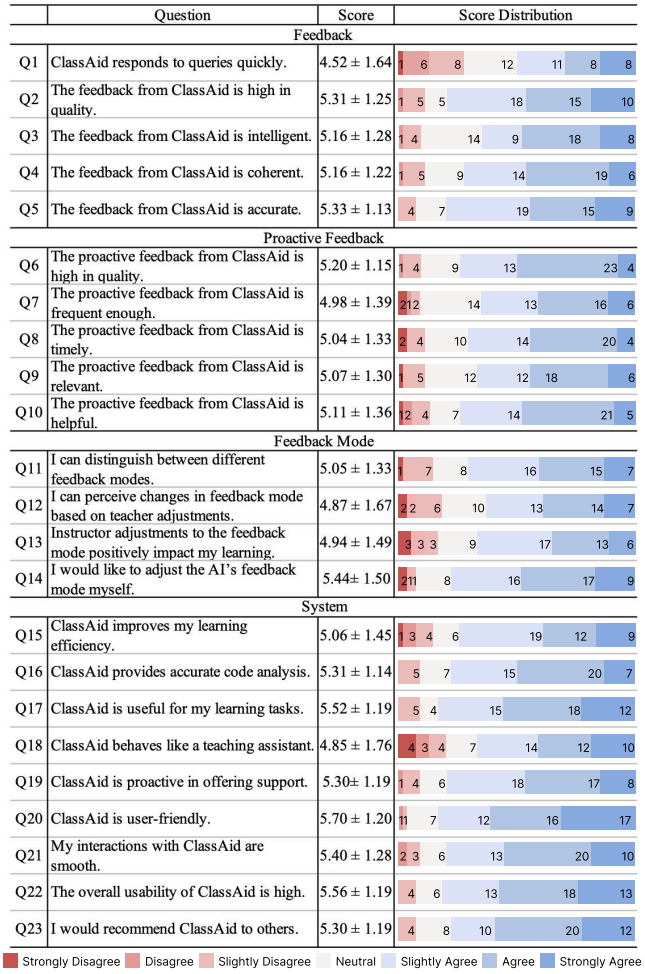

Figure 6: Student ratings on the *ClassAid*.

5.2 TA Agents' Cognitive-Level Assessment and Feedback Quality

To mitigate potential trust issues associated with LLM-based feedback and better understand how students interact with the TA Agent [29], we conducted expert evaluations of four components: (1) the accuracy of the Agent's Bloom-based estimation of students' cognitive levels, (2) the correctness of its feedback across modes, (3) the appropriateness of its feedback-type selection in Auto mode, and (4) the accuracy of its question-type classification (critical thinking vs. answer seeking). We drew a simple random sample comprising approximately 50% of all student questions collected in the user study (n = 274). Following StuGPTVis [9], two instructors with data visualization teaching experience (I1–I2) independently evaluated the accuracy of the Agent's Bloom-level classification for each question and coded each piece of feedback and question type using a three-point scale (1 = mostly correct, 0.5 = partially correct, 0 = mostly incorrect). To assess the Agent's ability to emulate instructor decision-making in Auto mode, we drew an additional simple random sample comprising about 50% of Auto-mode questions (n = 106), and I1 and I2 applied the same scale to rate the appropriateness of the selected feedback type.

5.3 Educator Interview

To gather additional feedback and improve the generalizability of our findings, we invited eight university-level programming educators (E1–E8) to participate in semi-structured interviews following the user study.

5.3.1 Participants. Participants were recruited through snowball sampling within the authors' professional networks [14]. All were actively teaching programming-related courses. The sample included five assistant professors (E1–E2, E4–E6) and three full professors (E3, E7–E8), with a gender distribution of three female and five male. Their teaching experience ranged from 1 to 23 years (M = 7.1). E1 and E3 had participated in the formative study, while the remaining participants were newly recruited. Course topics included data visualization (4), Python programming (2), and Java programming (2). Each educator received a $50 honorarium.

5.3.2 Procedure. Each interview (90 minutes) began with an introduction to the study's goals, a system overview, and guidance on using *ClassAid*. Participants then conducted an brief free exploration to familiarize themselves with both the student interface

and the instructor dashboard. To help participants experience the system in a context closer to real classroom use, we selected three segments from the user study data that ended at the timestamps where the instructor initiated high-frequency mode adjustments. These segments captured the student–AI interaction patterns that immediately preceded those adjustments, and each segment included the complete interaction records visible to instructors during the user study. Participants examined the three segments in sequence. For each segment, they reviewed the interaction logs, observed students' learning states, analyzed their difficulties, and used a think-aloud protocol to adjust TA Agent modes, explaining the reasoning behind their decisions. After each segment, we showed the corresponding classroom video for comparison with real instructional decisions. The interview concluded with a discussion of design goals, usability, limitations, and suggestions. All interviews were conducted via Zoom and audio recorded with consent.

6 Results

6.1 TA Agents' Cognitive-Level Assessment and Feedback Quality

This section summarizes the quantitative evaluation results of the TA Agent based on instructors' ratings across several dimensions.

6.1.1 Accuracy of of Cognitive-Level. We evaluated the accuracy of the TA Agent's cognitive-level predictions by randomly sampling 274 student questions. For each question, the Agent produced a Bloom-level classification, and two instructors (I1, I2) labeled the correctness of that classification using a three-level rubric. As shown in Table 4, the results demonstrated 95.99% agreement which indicates high a high rater consistency and supports the reliability of the TA Agent's cognitive-level estimates.

6.1.2 Correctness of Feedback. We randomly sampled 274 student questions, each with one heuristic and one technical feedback response. Two instructors (I1, I2) independently rated all responses using a three-level rubric, as shown in Table 5. Overall agreement reached 94.71%. These results indicate strong inter-rater consistency and suggest that the TA Agent provides reliable feedback across both modes.

6.1.3 Appropriateness of Auto Mode Selections. To assess the TA Agent's feedback-selection in Auto mode, we analyzed 50% of automatically generated feedback instances (n = 106), including 66 technical and 40 heuristic responses (in Table 6). I1 and I2 independently rated each response using a predefined three-level rubric. The overall percent agreement was 88.68% , indicating high rater consistency and reliable evaluation of the TA Agent's feedback quality.

6.1.4 Accuracy of Question Type. Among the 274 student questions, the system classified 175 as critical thinking and 99 as direct answer-seeking. I1 and I2 evaluated the accuracy of these classifications using the same three-level rubric (Table 7). The overall agreement was 95.99%, indicating that the system can reliably distinguish between critical-thinking and direct answer-seeking questions.

Across all evaluated components, the TA Agent demonstrated consistently strong performance. It provided accurate assessments of students' learning states and generated high-quality responses. Although some subjectivity remained in Auto-mode feedback selection, particularly in borderline cases, this does not imply insufficient performance. In many situations, human instructors also differ in determining whether a student would benefit more from technical or heuristic guidance, because such pedagogical decisions do not have a single correct answer.

6.2 System Evaluation

In this study, we conducted a systematic analysis of system interactions from both the student and instructor perspectives.

6.2.1 Student-Side Performance and Feedback Analysis. Table 8 summarizes student performance, TA Agent triggers, and feedback across the two programming tasks. Students achieved accuracy scores of 3.87 and 3.97, with average completion times of 21 and 17 minutes. In total, they made 6,841 code edits, 1,102 code executions, and 82 pauses. The TA Agent generated 394 proactive triggers, and 28 passive triggers (except student request). Students submitted 547 questions, resulting in 1,107 feedback messages, including 731 heuristic responses, 165 technical responses, and 211 Auto-mode responses (132 technical and 79 heuristic). Students provided 29 likes and 24 dislikes, corresponding to rating frequencies of 5.5% and 3.3%. Despite receiving over 1,100 feedback messages, students only rated a small fraction, consistent with the "extreme feedback bias" [18], where moderate experiences are less likely to elicit reactions.

6.2.2 Instructor-Side Interventions and Feedback Mode Adjustment Strategies. At the start of the in-class programming activity, the instructor and two TAs collaboratively used the *ClassAid* instructor dashboard to monitor student progress and manage TA Agents. As previously introduced (in Sec. 4.3), the *ClassAid* supports three types of alerts. The system triggered 15 alerts (four agent, three process, and eight outcome alerts), of which the team addressed four agent, three process, and five code alerts, enabling timely interventions.

To manage feedback mode at scale, the instructional team made eight classroom-wide feedback mode adjustments based on student progress and system cues. Initially, all students were set to heuristic mode to encourage independent and critical thinking. However, after 10 minutes with no task completions, the team switched to auto mode to offer more dynamic support. When half of the students completed Task 1, those still struggling were transitioned to technical mode. Later, when students moved on to Task 2, the class reverted to heuristic mode to reinforce independent problem-solving. This adaptive strategy supported diverse learning needs in real time.

In addition to class-level adjustments, the instructor reviewed logs for 46 students, most commonly due to repeated answer-seeking behavior, system alerts, code issues, or low performance. Based on these reviews, 22 students had their feedback modes individually adjusted. For example, one student exhibiting persistent answer-seeking was switched to silent mode to reduce AI over-reliance. Finally, for students making little progress even under technical mode, instructors or TAs provided additional human support. This included identifying difficulties, advising on TA Agent

Table 4: Inter-rater agreement on the TA Agent's cognitive-level classification, including detailed rating distributions and overall reliability.

Cognitive-level	Rating Level	I1	I2	Agreement Count	Agreement Ratio
Overall (n = 274)	Correct (1)	255	249	247	95.99%
	Partially Correct (0.5)	12	16	10	
	Incorrect (0)	7	9	6	

Table 5: Inter-rater agreement between I1 and I2 across feedback types, including fine-grained ratings and overall reliability.

Feedback Type	Rating Level	I1	I2	Agreement Count	Agreement Ratio
Heuristic (n = 274)	Correct (1)	253	249	245	95.26%
	Partially Correct (0.5)	16	21	12	
	Incorrect (0)	5	4	4	
Technical (n = 274)	Correct (1)	263	258	250	94.16%
	Partially Correct (0.5)	6	12	4	
	Incorrect (0)	5	4	4	
Total	–	–	–	**519**	**94.71%**

Table 6: Inter-rater agreement between I1 and I2 for Auto-mode feedback responses, including detailed ratings and overall reliability.

Feedback Type	Rating Level	I1	I2	Agreement Count	Agreement Ratio
Heuristic (n = 40)	Correct (1)	36	35	34	90.00%
	Partially Correct (0.5)	2	3	1	
	Incorrect (0)	2	2	1	
Technical (n = 66)	Correct (1)	57	52	50	87.88%
	Partially Correct (0.5)	6	10	5	
	Incorrect (0)	3	4	3	
Total	–	–	–	**94**	**88.68%**

Table 7: Inter-rater agreement between I1 and I2 on thinking types, including fine-grained ratings and overall reliability.

Thinking Type	Rating Level	I1	I2	Agreement Count	Agreement Ratio
Critical Thinking (n=175)	Correct (1)	166	164	163	98.29%
	Partially Correct (0.5)	9	10	8	
	Incorrect (0)	0	1	1	
Direct Answer Seeking (n=99)	Correct (1)	91	87	85	91.92%
	Partially Correct (0.5)	3	5	3	
	Incorrect (0)	5	7	3	
Total	–	–	–	**267**	**95.99%**

Table 8: Summary of student performance, TA Agent triggers, and feedback across Task1 and Task2.

Measure	Task1	Task2	Total
Accuracy (1–5)	3.87	3.97	–
Completion Time (min)	21	17	–
Code Edits	3,538	3,003	6,841
Code Executions	674	428	1,102
Pauses	25	57	82
Predictive Triggers	75	63	138
Proactive Triggers	267	127	394
Passive Triggers	5	23	28
Questions Submitted	393	154	547
Feedback Received	740	367	1,107
Heuristic Feedback	517	214	731
Technical Feedback	101	64	165
Auto-mode Feedback	122	89	211 (132 Tech., 79 Heur.)
Feedback Likes	21	8	29
Feedback Dislikes	20	4	24
Rating Rate	5.5%	3.3%	–

use, or offering direct assistance. In total, five students received personalized in-person help during the session.

6.3 Evaluation on Student Interface and Instructor Dashboard

6.3.1 Feedback from Students' Perspective. As shown in Fig. 6, student ratings of feedback, proactive feedback, feedback modes, and overall system experience were collected on a 7-point Likert scale.

General feedback received positive evaluations, with average scores above 5.0 for quality, intelligence, coherence, and accuracy, except for response speed. These results indicate that students generally found the TA Agent's feedback effective, although some reported dissatisfaction with latency when timely support was needed.

Proactive feedback was rated positively overall, with quality receiving the highest rating (M = 5.20) and frequency the lowest (M = 4.98). Some students felt the feedback arrived too quickly and limited independent reflection, whereas beginners appreciated the timely guidance. For instance, one student initially tried to assign colors manually until the Agent suggested a more concise specification: `color: "field": "category", "type": "nominal"`. She reported being pleasantly surprised because the Agent detected her difficulty and offered timely support without being prompted.

Regarding feedback mode adjustment, most students could distinguish among the modes, noticed instructor-initiated changes, and believed these shifts affected their performance. In the preference survey, Auto mode and technical feedback were each preferred by 42.6% of students, whereas 14.8% preferred heuristic feedback. Interviews showed that students who preferred heuristic feedback appreciated its concise and readable style, as well as the space it provided for independent problem solving. In contrast, technical feedback was viewed as direct and efficient, offering specific code suggestions and concrete fixes, although some students worried it might encourage overreliance on the AI. Students prioritizing task completion often preferred technical feedback, whereas those choosing Auto mode valued its dynamic adaptation, especially when unsure which style suited them. We also examined students' willingness to adjust the feedback mode manually. The average rating was 5.44, indicating strong interest in having more control. Many students expressed a desire to select modes themselves when the system's responses did not meet their expectations. One student noted that instructors may struggle to track individual settings in large classes and suggested adding a request feature to allow students to formally ask for mode changes.

Regarding the overall system evaluation, all aspects received high scores except for personality, with friendliness rated highest (M = 5.70). Students found the interface clean, intuitive, and beginner-friendly. The lower rating for personality likely reflects the Agent's uniform and repetitive response style. Although students appreciated the content, the rigid formatting created a more robotic impression, which may have negatively influenced their perception of its personality.

In follow-up interviews, we asked students to compare *ClassAid* with other AI tools, such as ChatGPT. Eighty percent of the students reported that *ClassAid* was better suited for classroom learning contexts, whereas general-purpose chatbots were more efficient when quick answers were needed. Students' comparisons primarily focused on several key dimensions.

Response time and usage efficiency. Most students noted that general chatbots had a clear advantage in providing immediate answers, making them suitable for situations with time pressure. However, students perceived this efficiency as often coming at the cost of reduced instructional guidance during the learning process.

Feedback form and analytical quality. Approximately 70% of the students emphasized that during learning, they preferred heuristic or proactive feedback that supported understanding rather than direct solutions. Students generally perceived the feedback provided by *ClassAid* as placing greater emphasis on reasoning processes and problem decomposition, which better supported conceptual understanding. As S1 noted, *"During learning, I prefer hints rather than answers."*

Learning support and instructor visibility. When encountering learning difficulties, more than half of the students (approximately 60%) reported that they were more likely to turn to AI tools rather than directly approach instructors, primarily due to feelings of shyness or discomfort. Although general AI tools could address some immediate issues, multiple students pointed out that the lack of instructor oversight made it difficult for instructors to identify students' misconceptions or monitor their learning progress. In contrast, approximately 70% of the students viewed *ClassAid* as providing a better balance between learning support and accountability by offering AI assistance while remaining visible to instructors. S4 explained, *"When I encounter learning problems, it is not that I do not want my instructor to know, but asking directly feels intimidating."*

These findings suggest that compared with other AI tools, *ClassAid* is uniquely positioned to support educational goals by offering pedagogically aligned feedback while maintaining transparency for instructors.

6.3.2 Feedback from Instructor Participants in the User Study. The instructional team expressed highly positive feedback on *ClassAid*, especially praising its real-time visualization of individual progress and classroom dynamics. The instructor noted that, unlike in earlier classes where engagement was hard to monitor, *ClassAid* instantly showed who had started working, which she described as "a surprising and impressive moment."

By clearly displaying each student's progress and difficulties, the system significantly reduced the instructor's cognitive load, enabling her to focus on students who required support and address class-wide issues more effectively. In user study, for instance, an omitted explanation of the "scheme" concept led to widespread errors. The system quickly identified this pattern, prompting the instructor to provide timely clarification. The system also helped the instructor gain a clearer understanding of students, particularly those who consistently struggled. One notable case involved a student who was unable to complete the task even with technical mode enabled. Further investigation revealed that the student had no prior experience with AI tools and lacked effective strategies for interacting with the system. The instructor provided guidance on how to formulate questions for the Agent, and with support from both the instructor and the Agent, the student ultimately completed the task.

Regarding feedback mode adjustment, the instructor emphasized that the system enhanced her instructional agency. In one instance, a student repeatedly asked the Agent for direct answers, prompting the instructor to switch the student's mode to silent. When the student inquired about the change, it created an opportunity to discuss the value of independent thinking. This interaction not only encouraged student reflection but also reinforced the instructor's role in shaping learning behaviors in AI-supported classrooms.

6.3.3 Feedback from Educators. Based on a thematic analysis of interviews with eight educators [16], we found that they expressed a broadly positive attitude toward *ClassAid*, emphasizing its controllability, flexibility, and the benefits of dynamic orchestration. They believed the system helps align AI support with pedagogical goals, manage class progress more effectively, and reduce the burden of providing personalized feedback in programming-intensive settings. E1 noted that the ability to adjust the AI Agent in real time reinforces the instructor's central role and ensures that the system remains highly controllable and adaptable.

Feedback Quality. Although both assistant and full professors expected high-quality feedback, their tolerance for the TA Agent's limitations differed. Assistant professors were generally more tolerant of imperfect responses, whereas full professors showed lower tolerance. E1 noted that most students are familiar with tools such as ChatGPT and therefore understand that AI-generated feedback may contain inaccuracies. E6 added that instructors still need to remind students to view AI responses as references rather than authoritative answers. However, E3 argued that relying solely on prompt engineering is insufficient for reliably distinguishing heuristic from technical feedback in Auto mode. He suggested using large-scale fine-tuning or retrieval-augmented generation to improve decision accuracy and ensure more appropriate guidance. E8 further observed that Auto-mode decisions depend on the system's accumulated understanding of students' learning states. Early in a course, this information is limited, which may lead to unstable mode selection. He therefore recommended enabling Auto mode only after sufficient student data has been collected. Building on this perspective, E7 noted that expectations for feedback quality depend on the type of question. Errors are unacceptable for problems with standardized answers, whereas flexible feedback is appropriate for open-ended problems.

System Usage. In contrast, assistant professors focused more on *ClassAid*'s visual monitoring capabilities. E7 praised the layout and information presentation of the instructor dashboard, noting that thoughtful visualization design reduced the burden of classroom monitoring. E6 observed that although the dashboard presents task scores clearly, richer behavioral information is scattered across multiple detailed views. This fragmentation makes it difficult for instructors to form a coherent understanding of students' learning processes and increases the effort needed to make fine-grained adjustments during class. E4 added that although the dashboard includes a summary layer through bar charts of question and error types, integrating these summaries with the other views remains challenging. For example, while the system can highlight students who frequently request direct answers, identifying and summarizing the specific questions they posed still requires manual exploration. These observations suggest a need for tighter integration between summary-level analytics and detailed behavioral views to help instructors interpret student progress more efficiently and adjust the Agent accordingly.

Agent Control. Instructors were unsurprised by the low frequency of student feedback on the TA Agent, noting that students tend to prioritize task completion. E2 suggested focusing instead on the Agent's ability to assess its own effectiveness through students' behavior, which may provide a more reliable supervisory mechanism. E5 further recommended using the relationship between code and feedback as an indirect indicator of the Agent's performance. When we proposed giving students limited control over their Agent mode, E3 raised concerns about potential misuse, such as repeatedly choosing technical mode to obtain answers. He suggested that usage restrictions would be necessary if the feature were implemented.

7 Discussion

7.1 Design Implications

Structured Design of Multi-stage Pedagogical Agents. *ClassAid* introduces a six-stage intelligent pedagogical agent grounded in formative and dynamic assessment theories [4, 48], as shown in Fig. 3. By abstracting how human instructors provide classroom feedback, the design aligns with the generative process of LLMs. This approach improves feedback quality and enhances the interpretability of what is typically a 'black box' in LLM-based responses [33]. Our findings highlight the value of refining and modularizing the individual levels of such agents. In the current design, only the *Consider* level allows instructor intervention, while the remaining levels follow fixed rules. Expanding configurability across multiple levels could give instructors finer control over how feedback is generated, including how student issues are identified, reviewed, and selected, thereby supporting more precise management of pacing and granularity. Such flexibility enables more personalized and context-sensitive learning experiences rather than relying on a uniform agent design.

Real-Time Class-Level Oversight through Feedback Monitoring. While recent research has explored student behavior tracking and feedback generation [65], mechanisms for aggregating these behaviors into actionable class-level insights remain limited, especially in in-person classroom settings [47]. Our observations suggest that summarizing irregular student behaviors into interpretable visual feedback can help educators quickly identify issues and intervene in a timely manner. In our implementation, these summaries supported monitoring at both the individual and class levels. Future systems may extend this approach by incorporating multi-layered visualizations that capture behavioral patterns across varying degrees of granularity and time, enabling more informed and responsive classroom decision-making.

Instructor-AI-Student Triangular Supervision of AI Agents. Although the potential of LLMs in education is widely acknowledged [55], their real-world deployment requires external oversight to ensure appropriate and accountable use [73]. One promising direction is a triangular supervision framework in which instructors and students jointly monitor and regulate the AI agent's feedback. Our implementation shows that such collaborative oversight can enhance transparency and foster trust in AI-supported instruction. Looking ahead, systems might incorporate self-supervised capabilities that allow AI agents to assess the effectiveness of their feedback based on student responses and adapt in real time to improve in-class performance.

7.2 Future Work

Fine-Grained Feedback for Personalized Learning Support. By integrating heuristic and technical feedback modes, *ClassAid* has shown promise in reducing students' overreliance on AI and limiting direct answer-seeking behavior. However, as students' needs become more nuanced, current feedback strategies remain too coarse-grained to fully support scaffolded learning pathways grounded in theories such as the Zone of Proximal Development [59] and instructional scaffolding [2]. Future work should explore fine-grained learner modeling to support tiered TA Agent responses. For example, within the technical feedback mode, systems should distinguish among sample code, pseudocode, and full solutions, allowing the depth of feedback to adapt dynamically to students' proficiency levels [32]. Additionally, instructional tasks vary in their pedagogical objectives, underscoring the need to align feedback content with specific learning goals.

Exploring Shared Control Mechanisms Between Instructors and Students. Managing TA Agent modes becomes increasingly burdensome in larger classroom settings, where untimely adjustments may compromise both learning experiences and the quality of personalized feedback. Students who actively regulate their learning may benefit from limited control over the agent [5, 77], but granting full control could raise instructors' supervisory load [26]. Students also vary widely in their capacity to manage such control effectively, depending on their educational background and prior experience with AI-supported environments [38]. A more balanced approach would allow students to request permissions or make a small number of adjustments, or rely on stronger self-supervision mechanisms within the agent, thereby supporting autonomy without increasing instructor workload.

Integrating Multi-Source Learning Data to Broaden System Applicability. The future development of TA Agents should extend beyond prompt engineering by incorporating diverse learning data such as student history, behavioral traces, and performance patterns [80]. Techniques such as fine-tuning and retrieval-augmented generation (RAG) can further enhance the agent's ability to interpret educational contexts [39]. To support a broader range of courses, a modular configuration system is also needed, one that enables automatic updates to knowledge and prompt libraries based on course content. Finally, allowing instructors to fine-tune the agent's feedback according to specific learning objectives will enhance the relevance and flexibility of AI support across diverse classroom environments.

7.3 Limitation

Scalability The current deployment involved 54 students in a single session, but larger-scale or extended deployments may introduce additional challenges. As classroom size increases, system response time, server load, and the instructor's capacity to monitor more student–AI interactions may become bottlenecks. Managing TA Agent modes for many students could also increase instructor workload and reduce the timeliness of pedagogically meaningful adjustments.

Multi-session and higher-enrollment studies are needed to evaluate scalability and identify potential constraints in long-term or institution-wide use.

Generalizability Our study examined how the TA Agent supported students in completing relatively simple Vega-Lite tasks. Thus, it remains unclear how well the system would generalize to more complex programming environments, diverse toolchains, or other pedagogical contexts. The study was also limited to a single in-person session with one student group and no longitudinal observation, restricting insights into long-term stability and broader applicability. Although the deployment focused on an in-person setting, future work should explore online or hybrid classrooms, where real-time orchestration may be especially valuable. Broader evaluations across subject areas, task complexity, teaching modalities, and learner profiles are needed to assess the system's generalizability.

8 Conclusion

This paper introduces *ClassAid*, a real-time classroom orchestration system that integrates an intelligent TA Agent and an instructor dashboard to facilitate responsive feedback and instructor-AI collaboration. At its core is a six-stage framework that enables the TA Agent to monitor student behavior, infer cognitive states, select feedback strategies, and deliver targeted interventions to support metacognitive development. We deployed *ClassAid* in a graduate-level programming course (n = 54) and found that the TA Agent provided accurate and personalized feedback, while the instructor dashboard enabled real-time oversight and dynamic control of feedback modes. Together, these features supported instructors in maintaining pedagogical authority, addressing emerging learning needs, and reducing the cognitive load of managing large classrooms. Quantitative results demonstrate the TA Agent's strong performance in feedback generation and decision-making, and qualitative feedback from students and educators highlights the system's usability, instructional value, and potential for responsible AI integration. Overall, this study proposes a novel instructor-AI collaboration model for programming classrooms, offering a scalable, adaptive solution to the challenges of real-time feedback and classroom orchestration in the era of generative AI.

Acknowledgments

Guodao Sun and Meng Xia are the corresponding authors. The work was supported by the National Natural Science Foundation of China, (62422607, 62372411, 62432014) and the Zhejiang Provincial Natural Science Foundation of China (QKWL25F0301).

References

[1] Sarah W Beck and Sarah R Levine. 2023. Backtalk: ChatGPT: A powerful technology tool for writing instruction. *Phi Delta Kappan* 105, 1 (2023), 66–67.
[2] Brian R Belland. 2017. *Instructional scaffolding in STEM education: Strategies and efficacy evidence.* Springer Nature.
[3] Hugh H Benson. 2011. Socratic method. *The Cambridge companion to socrates* (2011), 179–200.
[4] Paul Black and Dylan Wiliam. 2009. Developing the theory of formative assessment. *Educational Assessment, Evaluation and Accountability (formerly: Journal of personnel evaluation in education)* 21 (2009), 5–31.
[5] Conrad Borchers, Jeroen Ooge, Cindy Peng, and Vincent Aleven. 2025. How Learner Control and Explainable Learning Analytics About Skill Mastery Shape Student Desires to Finish and Avoid Loss in Tutored Practice. In *Proceedings of the 15th International Learning Analytics and Knowledge Conference*. 810–816.
[6] Douglas Carnine, Jerry Silbert, Edward J Kameenui, and Sara G Tarver. 1997. *Direct instruction reading*. Merrill Columbus, OH.
[7] John Chen, Xi Lu, Yuzhou Du, Michael Rejtig, Ruth Bagley, Mike Horn, and Uri Wilensky. 2024. Learning agent-based modeling with LLM companions: Experiences of novices and experts using ChatGPT & NetLogo chat. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–18.
[8] Valerie Chen, Alan Zhu, Sebastian Zhao, Hussein Mozannar, David Sontag, and Ameet Talwalkar. 2024. Need Help? Designing Proactive AI Assistants for Programming. *arXiv preprint arXiv:2410.04596* (2024).
[9] Zixin Chen, Jiachen Wang, Meng Xia, Kento Shigyo, Dingdong Liu, Rong Zhang, and Huamin Qu. 2024. StuGPTViz: A Visual Analytics Approach to Understand Student-ChatGPT Interactions. *IEEE Transactions on Visualization and Computer Graphics* (2024).
[10] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C Desmarais, and Zhen Ming Jack Jiang. 2023. Github copilot ai pair programmer: Asset or liability? *Journal of Systems and Software* 203 (2023), 111734.
[11] Paul Denny, Juho Leinonen, James Prather, Andrew Luxton-Reilly, Thezyrie Amarouche, Brett A Becker, and Brent N Reeves. 2023. Promptly: Using prompt problems to teach learners how to effectively utilize ai code generators. *arXiv preprint arXiv:2307.16364* (2023).
[12] Iria Estévez-Ayres, Patricia Callejo, Miguel Ángel Hombrados-Herrera, Carlos Alario-Hoyos, and Carlos Delgado Kloos. 2024. Evaluation of LLM Tools for Feedback Generation in a Course on Concurrent Programming. *International Journal of Artificial Intelligence in Education* (2024), 1–17.
[13] Mary Forehand. 2010. Bloom's taxonomy. *Emerging perspectives on learning, teaching, and technology* 41, 4 (2010), 47–56.
[14] Lisa M Given. 2008. *The Sage encyclopedia of qualitative research methods*. Sage publications.
[15] Elena L Glassman, Jeremy Scott, Rishabh Singh, Philip J Guo, and Robert C Miller. 2015. OverCode: Visualizing variation in student solutions to programming problems at scale. *ACM Transactions on Computer-Human Interaction (TOCHI)* 22, 2 (2015), 1–35.
[16] Greg Guest, Kathleen M MacQueen, and Emily E Namey. 2011. *Applied thematic analysis*. sage publications.
[17] Philip J Guo. 2015. Codeopticon: Real-time, one-to-many human tutoring for computer programming. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. 599–608.
[18] Kent D Harber. 1998. Feedback to minorities: Evidence of a positive bias. *Journal of personality and social psychology* 74, 3 (1998), 622.
[19] Emma Harvey, Allison Koenecke, and Rene F Kizilcec. 2025. " Don't Forget the Teachers": Towards an Educator-Centered Understanding of Harms from Large Language Models in Education. *arXiv preprint arXiv:2502.14592* (2025).
[20] John Hattie and Helen Timperley. 2007. The power of feedback. *Review of educational research* 77, 1 (2007), 81–112.
[21] Andrew Head, Elena Glassman, Gustavo Soares, Ryo Suzuki, Lucas Figueredo, Loris D'Antoni, and Björn Hartmann. 2017. Writing reusable code feedback at scale with mixed-initiative program synthesis. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*. 89–98.
[22] George E Hein. 1991. Constructivist learning theory. *Institute for Inquiry* 14 (1991).
[23] Cindy E Hmelo-Silver. 2004. Problem-based learning: What and how do students learn? *Educational psychology review* 16, 3 (2004), 235–266.
[24] Kenneth Holstein, Bruce M McLaren, and Vincent Aleven. 2019. Co-designing a real-time classroom orchestration tool to support teacher-AI complementarity. *Grantee Submission* (2019).
[25] Kenneth Holstein, Bruce M McLaren, and Vincent Aleven. 2019. Designing for complementarity: Teacher and student needs for orchestration support in AI-enhanced classrooms. In *Artificial Intelligence in Education: 20th International Conference, AIED 2019, Chicago, IL, USA, June 25-29, 2019, Proceedings, Part I 20*. Springer, 157–171.
[26] Ken Holstein and Jennifer K Olsen. 2023. Human-AI co-orchestration: the role of artificial intelligence in orchestration. In *Handbook of artificial intelligence in education*. Edward Elgar Publishing, 309–321.
[27] Xinying Hou, Zihan Wu, Xu Wang, and Barbara J Ericson. 2024. Codetailor: Llm-powered personalized parsons puzzles for engaging support while learning programming. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*. 51–62.
[28] Sven Jacobs and Steffen Jaschke. 2024. Evaluating the application of large language models to generate feedback in programming education. In *2024 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 1–5.
[29] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM computing surveys* 55, 12 (2023), 1–38.
[30] Breanna Jury, Angela Lorusso, Juho Leinonen, Paul Denny, and Andrew Luxton-Reilly. 2024. Evaluating llm-generated worked examples in an introductory programming course. In *Proceedings of the 26th Australasian computing education*

conference. 77–86.

[31] Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, et al. 2023. ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and individual differences* 103 (2023), 102274.

[32] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Zachary Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In *Proceedings of the 2024 chi conference on human factors in computing systems*. 1–20.

[33] Hassan Khosravi, Simon Buckingham Shum, Guanliang Chen, Cristina Conati, Yi-Shan Tsai, Judy Kay, Simon Knight, Roberto Martinez-Maldonado, Shazia Sadiq, and Dragan Gasevic. 2022. Explainable artificial intelligence in education. *Computers and education: artificial intelligence* 3 (2022), 100074.

[34] Juho Kim, Elena L Glassman, Andrés Monroy-Hernández, and Meredith Ringel Morris. 2015. RIMES: Embedding interactive multimedia exercises in lecture videos. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. 1535–1544.

[35] Akit Kumar, MS Lakshmi Devi, and Jeffrey S Saltz. 2023. Bridging the gap in ai-driven workflows: The case for domain-specific generative bots. In *2023 IEEE International Conference on Big Data (BigData)*. IEEE, 2421–2430.

[36] Kimio Kuramitsu, Yui Obara, Miyu Sato, and Momoka Obara. 2023. Kogi: A seamless integration of ChatGPT into Jupyter environments for programming education. In *Proceedings of the 2023 ACM SIGPLAN International Symposium on SPLASH-E*. 50–59.

[37] Yu-Ju Lan and Nian-Shing Chen. 2024. Teachers' agency in the era of LLM and generative AI. *Educational Technology & Society* 27, 1 (2024), I–XVIII.

[38] LuEttaMae Lawrence, Vanessa Echeverria, Kexin Yang, Vincent Aleven, and Nikol Rummel. 2024. How teachers conceptualise shared control with an AI co-orchestration tool: A multiyear teacher-centred design process. *British Journal of Educational Technology* 55, 3 (2024), 823–844.

[39] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33 (2020), 9459–9474.

[40] Jian Liao, Linrong Zhong, Longting Zhe, Handan Xu, Ming Liu, and Tao Xie. 2024. Scaffolding computational thinking with ChatGPT. *IEEE Transactions on Learning Technologies* (2024).

[41] Mark Liffiton, Brad E Sheese, Jaromir Savelka, and Paul Denny. 2023. Codehelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. 1–11.

[42] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J Malan. 2024. Teaching CS50 with AI: leveraging generative artificial intelligence in computer science education. In *Proceedings of the 55th ACM technical symposium on computer science education V. 1*. 750–756.

[43] Xingyu Bruce Liu, Shitao Fang, Weiyan Shi, Chien-Sheng Wu, Takeo Igarashi, and Xiang Anthony Chen. 2024. Proactive Conversational Agents with Inner Thoughts. *arXiv preprint arXiv:2501.00383* (2024).

[44] Michelle Lui, Kit-Ying Angela Chong, Martha Mullally, and Rhonda McEwen. 2023. Facilitated model-based reasoning in immersive virtual reality: Meaning-making and embodied interactions with dynamic processes. *International Journal of Computer-Supported Collaborative Learning* 18, 2 (2023), 203–230.

[45] Wenhan Lyu, Yimeng Wang, Tingting Chung, Yifan Sun, and Yixuan Zhang. 2024. Evaluating the effectiveness of llms in introductory computer science education: A semester-long field study. In *Proceedings of the Eleventh ACM Conference on Learning@ Scale*. 63–74.

[46] Roberto Martinez-Maldonado, Andrew Clayphan, Kalina Yacef, and Judy Kay. 2014. MTFeedback: providing notifications to enhance teacher awareness of small group work in the classroom. *IEEE Transactions on Learning Technologies* 8, 2 (2014), 187–200.

[47] Jennifer Meyer, Thorben Jansen, Ronja Schiller, Lucas W Liebenow, Marlene Steinbach, Andrea Horbach, and Johanna Fleckenstein. 2024. Using LLMs to bring evidence-based feedback into the classroom: AI-generated feedback increases secondary students' text revision, motivation, and positive emotions. *Computers and Education: Artificial Intelligence* 6 (2024), 100199.

[48] Norris Minick. 1987. Implications of Vygotsky's theories for dynamic assessment. (1987).

[49] Reza Hadi Mogavi, Chao Deng, Justin Juho Kim, Pengyuan Zhou, Young D Kwon, Ahmed Hosny Saleh Metwally, Ahmed Tlili, Simone Bassanelli, Antonio Bucchiarone, Sujit Gujar, et al. 2024. ChatGPT in education: A blessing or a curse? A qualitative study exploring early adopters' utilization and perceptions. *Computers in Human Behavior: Artificial Humans* 2, 1 (2024), 100027.

[50] Dylan Edward Moore, Sophia RR Moore, Bansharee Ireen, Winston P Iskandar, Grigory Artazyan, and Elizabeth L Murnane. 2024. Teaching artificial intelligence in extracurricular contexts through narrative-based learnersourcing. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–28.

[51] Kenneth D Moore. 2014. *Effective instructional strategies: From theory to practice*. Sage Publications.

[52] Andrés Neyem, Luis A González, Marcelo Mendoza, Juan Pablo Sandoval Alcocer, Leonardo Centellas, and Carlos Paredes. 2024. Towards an AI knowledge assistant for context-aware learning experiences in software capstone project development. *IEEE Transactions on Learning Technologies* (2024).

[53] Sydney Nguyen, Hannah McLean Babe, Yangtian Zi, Arjun Guha, Carolyn Jane Anderson, and Molly Q Feldman. 2024. How Beginning Programmers and Code LLMs (Mis) read Each Other. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–26.

[54] Omid Noroozi, Paul A Kirschner, Harm JA Biemans, and Martin Mulder. 2018. Promoting argumentation competence: Extending from first-to second-order scaffolding through adaptive fading. *Educational psychology review* 30, 1 (2018), 153–176.

[55] Hyanghee Park and Daehwan Ahn. 2024. The promise and peril of ChatGPT in higher education: opportunities, challenges, and design implications. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–21.

[56] Joon Sung Park, Carolyn Q Zou, Aaron Shaw, Benjamin Mako Hill, Carrie Cai, Meredith Ringel Morris, Robb Willer, Percy Liang, and Michael S Bernstein. 2024. Generative agent simulations of 1,000 people. *arXiv preprint arXiv:2411.10109* (2024).

[57] Kevin Pu, Daniel Lazaro, Ian Arawjo, Haijun Xia, Ziang Xiao, Tovi Grossman, and Yan Chen. 2025. Assistance or Disruption? Exploring and Evaluating the Design and Trade-offs of Proactive AI Programming Support. *arXiv preprint arXiv:2502.18658* (2025).

[58] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. https://vega.github.io/vega-lite/. Accessed: [Insert date you accessed the site].

[59] Karim Shabani, Mohamad Khatib, and Saman Ebadi. 2010. Vygotsky's zone of proximal development: Instructional implications and teachers' professional development. *English language teaching* 3, 4 (2010), 237–248.

[60] Valerie J Shute. 2008. Focus on formative feedback. *Review of educational research* 78, 1 (2008), 153–189.

[61] Robert L Solso, M Kimberly MacLin, and Otto H MacLin. 2005. *Cognitive psychology*. Pearson Education New Zealand.

[62] Rayne A Sperling, Bruce C Howard, Richard Staley, and Nelson DuBois. 2004. Metacognition and self-regulated learning constructs. *Educational research and evaluation* 10, 2 (2004), 117–139.

[63] John Sweller. 1988. Cognitive load during problem solving: Effects on learning. *Cognitive science* 12, 2 (1988), 257–285.

[64] Mei Tan and Hari Subramonyam. 2024. More than model documentation: uncovering teachers' bespoke information needs for informed classroom integration of ChatGPT. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–19.

[65] Xiaohang Tang, Sam Wong, Marcus Huynh, Zicheng He, Yalong Yang, and Yan Chen. 2024. SPHERE: Scaling Personalized Feedback in Programming Classrooms with Structured Review of LLM Outputs. *arXiv preprint arXiv:2410.16513* (2024).

[66] Xiaohang Tang, Sam Wong, Kevin Pu, Xi Chen, Yalong Yang, and Yan Chen. 2024. VizGroup: An AI-Assisted Event-Driven System for Real-Time Collaborative Programming Learning Analytics. *arXiv preprint arXiv:2404.08743* (2024).

[67] Anouschka van Leeuwen, Nikol Rummel, et al. 2019. Orchestration tools to support the teacher during student collaboration: a review. *Unterrichtswissenschaft* 47, 2 (2019), 143–158.

[68] Kurt VanLehn. 2011. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational psychologist* 46, 4 (2011), 197–221.

[69] Kurt VanLehn, Hugh Burkhardt, Salman Cheema, Daniel Pead, Alan Schoenfeld, and Jon Wetzel. 2018. How can FACT encourage collaboration and self-correction? In *Deep Comprehension*. Routledge, 114–127.

[70] April Yi Wang, Yan Chen, John Joon Young Chung, Christopher Brooks, and Steve Oney. 2021. Puzzleme: Leveraging peer assessment for in-class programming exercises. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (2021), 1–24.

[71] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682* (2022).

[72] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[73] Ben Williamson, Alex Molnar, and Faith Boninger. 2024. Time for a Pause: Without Effective Public Oversight, AI in Schools Will Do More Harm Than Good. *Commercialism in Education Research Unit* (2024).

[74] Juliette Woodrow, Ali Malik, and Chris Piech. 2024. Ai teaches the art of elegant coding: Timely, fair, and helpful style feedback in a global course. In *Proceedings*

of the 55th ACM Technical Symposium on Computer Science Education V. 1. 1442–1448.

[75] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 2022. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In *Proceedings of the 2022 CHI conference on human factors in computing systems*. 1–22.

[76] Lixiang Yan, Lele Sha, Linxuan Zhao, Yuheng Li, Roberto Martinez-Maldonado, Guanliang Chen, Xinyu Li, Yueqiao Jin, and Dragan Gašević. 2024. Practical and ethical challenges of large language models in education: A systematic scoping review. *British Journal of Educational Technology* 55, 1 (2024), 90–112.

[77] Kexin Bella Yang, Vanessa Echeverria, Zijing Lu, Hongyu Mao, Kenneth Holstein, Nikol Rummel, and Vincent Aleven. 2023. Pair-up: prototyping human-AI co-orchestration of dynamic transitions between individual and collaborative learning in the classroom. In *Proceedings of the 2023 CHI conference on human factors in computing systems*. 1–17.

[78] Kexin Bella Yang, LuEttaMae Lawrence, Vanessa Echeverria, Boyuan Guo, Nikol Rummel, and Vincent Aleven. 2021. Surveying teachers' preferences and boundaries regarding human-AI control in dynamic pairing of students for collaborative learning. In *Technology-Enhanced Learning for a Free, Safe, and Sustainable World: 16th European Conference on Technology Enhanced Learning, EC-TEL 2021, Bolzano, Italy, September 20-24, 2021, Proceedings 16*. Springer, 260–274.

[79] Yinuo Yang, Ashley Ge Zhang, Steve Oney, and April Yi Wang. 2025. SPARK: Real-Time Monitoring of Multi-Faceted Programming Exercises. (2025).

[80] Dan Ye and Svoboda Pennisi. 2022. Using trace data to enhance Students' self-regulation: A learning analytics perspective. *The Internet and Higher Education* 54 (2022), 100855.

[81] Ashley Ge Zhang, Yan Chen, and Steve Oney. 2023. Vizprog: Identifying misunderstandings by visualizing students' coding progress. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–16.

[82] Gefei Zhang, Shenming Ji, Yicao Li, Jingwei Tang, Jihong Ding, Meng Xia, Guodao Sun, and Ronghua Liang. 2025. CPVis: Evidence-based Multimodal Learning Analytics for Evaluation in Collaborative Programming. *arXiv preprint arXiv:2502.17835* (2025).

[83] Lanqin Zheng, Yuanyi Zhen, Jiayu Niu, and Lu Zhong. 2022. An exploratory study on fade-in versus fade-out scaffolding for novice programmers in online collaborative programming settings. *Journal of Computing in Higher Education* 34, 2 (2022), 489–516.

A Formative Study

A.1 Participant Demographics

Table 9: Instructor Demographics and Classroom Information

ID	Gender	Age	Experience (Years)	Class Size	No. of Activities
T1	Male	36	8	40	5
T2	Male	29	5	50	4
T3	Female	49	23	100	8
T4	Female	33	6	45	5
T5	Male	36	6	100	10
T6	Female	34	5	100	5
T7	Female	31	4	70	4

A.2 Interview Question for Instructor

1. Basic Information

(1) What is your age and gender?
(2) How many years of programming teaching experience do you have?
(3) How many students are typically in your class?
(4) How many programming practice sessions do you usually organize during a course?

2. Understanding Your Teaching Experience

(5) In programming courses, do you prefer students to complete lab assignments individually or in groups?
(6) Why do you prefer individual lab work or group lab work?
Follow-up: What do you think are the advantages and disadvantages of individual lab work?
Advantages for students and instructors:
Disadvantages for students and instructors:

3. Barriers to Teacher Student Interaction

(7) Do you observe any barriers that make students reluctant to ask questions, such as social distance or fear of asking simple questions?
Follow-up: Have you found ways to address this, or is it difficult to change?

4. Instructional Guidance

(8) How do you usually guide students during programming activities? For example, prompting them to reason or giving direct solutions? Could you provide an example?
(9) How do you decide the level of guidance to provide when a student asks for help?
Follow-up: Does your decision depend on student performance, time constraints, or other factors?
(10) Besides giving concrete solutions and offering heuristic encouragement, do you have other ways of guiding students?
(11) Do you proactively offer help during labs? What factors influence your decision to intervene?

5. During Student Agent Interactions

(12) If an AI agent provided feedback to students, what capabilities or types of feedback would you want it to have?

(13) What information about student–agent interactions would you want access to (e.g., conversation logs, task progress, question types)?
(14) Do you think it is necessary to monitor or evaluate the agent's performance? How would you assess whether it is helping students effectively?
(15) Do you need information about the students' task completion status?
(16) In general, do you believe an LLM-based agent can support student learning effectively?

6. Perspectives on Agents

(17) If an AI agent could monitor or participate in student discussions, what functions or characteristics would you want it to have?
(18) What aspects of the agent's behavior should be adjustable, and who should have control (instructors, students, or both)?
(19) Would you prefer the agent to actively participate in student collaboration or to provide feedback only when needed?
(20) What aspects of the agent's behavior should be adjustable, and who should have control (instructors, students, or both)?
(21) Do you trust the outputs of LLM-based agents? Under what conditions would you need explanations or insights from them (e.g., participation analysis, concept understanding, off-topic detection)?

B Task Design

We deployed ClassAid in a graduate-level Data Visualization course offered by the Department of Computer Science at the local research university. The course combines theoretical instruction with in-class programming activities to introduce key principles of data visualization and their practical implementation. To prepare for deployment, we met with the course instructor over three one-hour online meetings. After discussing the feasibility of integration, the instructor agreed to adopt ClassAid for in-class use, and we collaboratively defined the classroom programming tasks.

B.1 Task Design

Both Task 1 and Task 2 used the same synthetic dataset consisting of 100 two-tuples, where one element is a numeric score and the other is a categorical label. The dataset exhibited clusters of values within specific ranges, making it particularly suitable for binned statistics and aggregation-based visual analysis. The tasks were designed to help students uncover score distributions and explore relationships between categories.

B.1.1 Task 1: Score Distribution Chart. In Task 1, students were asked to use Vega-Lite to create a bar chart showing the distribution of scores across predefined ranges. The x-axis was required to display binned score intervals, while the y-axis represented the count of scores within each bin. Additionally, appropriate axis labels and a chart title were expected. This task assessed students' ability to create basic bar charts using Vega-Lite, focusing on binning continuous variables, aggregating values per bin, and presenting the results in a readable format.

B.1.2 Task 2: Class Average Scores. Task 2 required students to create a bar chart depicting the average score for each category (A, B, C, D, E). The x-axis represented categories, the y-axis displayed average scores, and bars were colored based on category. Building on the first task, Task 2 evaluated students' ability to perform data aggregation, assign categorical values to the x-axis, apply color encoding, and present average values in a clear and interpretable chart.

C Interview Questions for Students

1. Prior Experience with AI Tools

(1) Have you used AI tools to assist with programming before? How about during classroom programming activities?
(2) What motivated you to use AI? Was it mainly to get direct answers?
(3) Did you find the AI-generated responses helpful?
(4) Do you feel that relying on AI allowed you to complete the task without truly learning?
(5) Have you used AI tools in other classes? Were those uses allowed by your instructors?

2. Comparing Our System with Other AI Tools

(6) When using our system, did you interact with it the same way you would with other AI tools (e.g., directly asking for answers or requesting code fixes)?
(7) Did you notice any differences between our system's responses and those from other AI tools? What were they?
(8) Which type of response did you prefer, and why?

3. Social and Emotional Reactions

(9) Did the fact that teachers could view your AI interactions make you feel uncomfortable or less willing to ask direct questions?

4. System Usage Patterns and Feedback Perception

(10) Did you frequently check the provided tutorials? Did you rely more on the AI or the tutorials to complete tasks?
(11) How would you evaluate the AI responses in our system? What were the strengths and weaknesses?
(12) What had the greatest impact on your experience? Do you have any examples to share?
(13) When you received proactive feedback, did it arrive in a timely and useful manner? Why or why not?

5. Feedback Evaluation and AI Mode Awareness

(14) When did you feel inclined to rate the AI's feedback? Or why did you choose not to rate it?
(15) In which cases were you most likely to give a rating (e.g., when the feedback was very wrong or particularly helpful)?
(16) Did you notice when the teacher adjusted your AI mode? Could you guess why? Was the adjustment timely?
(17) Did you perceive differences between the AI modes? If so, what were they?
(18) Which mode did you prefer, and why?
(19) If you could choose your own AI mode, what factors would guide your decision?

6. Learning Outcomes and Teacher Interaction

(20) Compared to before, do you feel you gained more knowledge or had a better grasp of the material?

(21) Did the system enhance your interaction with the teacher? Besides adjusting the AI, did the teacher provide any extra support?

7. Reflections and Future Expectations

(22) Do you have any suggestions for improving the system?
(23) Would you prefer the system to help you expand your knowledge or simply assist in completing the task?
(24) If the system were applied to other subjects such as Python learning, would you be willing to use it?

D Interview Questions for Instructors

1. Teaching Practices and Existing Challenges

(1) How did you typically organize programming activities in your previous classes? What challenges did you face?
(2) Were you able to monitor students' learning behaviors and progress in real time? Did they actively ask for help?
(3) Were you previously aware of students' weaknesses in specific areas? How did you evaluate their programming skills then, and how does that compare to now?

2. System Impact and Perceived Changes

(4) Do you think our system has helped address some of the issues you faced before? Could you elaborate?
(5) Has the system helped you better understand students' programming abilities and individual differences?
(6) Were you able to identify students who needed additional support beyond the AI? What actions did you take?
(7) Besides adjusting the AI feedback mode, did you have any other interactions with students?
(8) Which part of the system did you find most useful? Why?
(9) Which part did you find less useful? Why?

3. Information Presentation and System Performance

(10) Do you find the system's information sufficiently detailed? Were there any insights you wished to see but couldn't?
(11) Did this information help create more opportunities for interacting with students?
(12) Was the data visualization intuitive and easy to interpret?
(13) How would you evaluate the system's responsiveness and real-time performance?
(14) Did you find the system workflow smooth? Was it overly complex at any point?

4. AI Feedback Quality and Control Strategy

(15) From your observation, was the AI-generated feedback helpful to students? How would you rate its quality? Did it help students complete tasks or improve their coding skills?
(16) Was the system's *proactive feedback* effective? Did you observe students' attitudes toward it? Were there students who disliked it, or none who explicitly liked it?
(17) What was your strategy for adjusting the AI feedback modes? Were there general rules or specific student behaviors that guided your decisions?

5. Notable Events and Student Reactions

(18) Were there any interesting or memorable events during your use of the system? (e.g., the student in Silent Mode who still struggled even after switching to Technical Mode)
(19) Did you receive any direct feedback from students? What did they say?

6. Teaching Load and Trust in AI

(20) Did the system help reduce your teaching burden? For example, by simplifying feedback delivery or improving awareness of student performance?
(21) After being given control over the AI, did you feel more confident or trusting in its role?

7. Insights and Reflections

(22) Did you discover anything new through using the system? For example, students not knowing how to ask questions or struggling to articulate their problems to the AI?
(23) Would you be willing to continue using the system in the future? Why or why not?

E Prompt for Reviewing and Assessing Student History

```
prompt = """
You are a ReviewlAgent responsible for reviewing and summarizing a student's learning trajectory in response to system triggers. Your goal is to assess the student's current and past progress to support the delivery of targeted, pedagogically aligned feedback.

You will receive:
- Recent activity context: includes latest interactions, code submissions, and AI feedback traces.
- Historical learning profile: includes completed tasks, concept mastery, performance patterns, and learner preferences.

Your job is to produce a structured summary across five dimensions:
1. **Cognitive Analysis**: Determine the current Bloom level and confidence trend based on recent interactions. Identify signs of stagnation or regression. Use Bloom's taxonomy to classify student questions or actions into one of six levels: Remember, Understand, Apply, Analyze, Evaluate, Create. Include reasoning for the classification.
2. **Error Analysis**: Extract error types, frequency, and distribution. Identify recurring mistakes and infer potential conceptual misunderstandings.
3. **Learning History**: Report preferred feedback mode, completed tasks, success rate, and learning style.
4. **Current State**: Assess current task status, recent triggers, activity level, and code quality.
5. **Knowledge State**: Distinguish mastered vs. struggling concepts, and highlight areas needing further support.

--- Input Schema ---
{
  "recent_activity": { ... },
  "historical_profile": { ... }
}

--- Output Format ---
{
  "cognitive_analysis": {
    "level": "apply",
    "confidence": 0.8,
```

```
    "reasoning": "The student is asking how to apply a specific encoding to a Vega-Lite chart, which aligns with the Apply level."
  },
  "error_analysis": { ... },
  "learning_history": { ... },
  "current_state": { ... },
  "knowledge_state": { ... },
  "metadata": { "is_auto_generated": true }
}

Behavioral Rules:
- Use Bloom's taxonomy to infer and track cognitive progression.
- Identify stagnation if levels drop or remain unchanged.
- Analyze code execution success and progress to infer task phase.
- Detect high-frequency error patterns linked to conceptual gaps.
- Maintain structured, valid output suitable for downstream reasoning agents.
"""
```

F Prompt for Considering Appropriate Forms of Learning Support

```
    """
You are a ThoughtFormation agent that transforms user input, student state, and code context
into structured 'thoughts' for downstream tutoring agents. You generate multiple categorized
responses to support reflective feedback, scaffolded guidance, and proactive corrections.

Your job is to segment the incoming data into actionable thinking paths for three feedback agents:
1. TechnicalAgent (for code-based explanation and fixes)
2. HeuristicAgent (for reflective, question-based prompts)
3. MetaAgent (for cognitive-level insights and metadata packaging)

**Input Schema**:
{
    "user_message": "Why is my chart not showing any bars?",
    "current_code": "{ \"mark\": \"bar\", \"data\": {}, \"encoding\": {} }",
    "response_mode": "auto",
    "retrieval_result": {
        "cognitive_analysis": {
            "level": "understand",
            "confidence": 0.8
        },
        "error_analysis": {
            "patterns": [
                { "type": "data", "message": "Missing values field" }
            ],
            "most_common": "data"
        },
        "learning_history": {
            "preferred_mode": "technical",
            "completed_tasks_count": 4
        },
        "current_state": {
            "recent_triggers": [{"type": "run", "is_auto_generated": true}]
        },
        "metadata": {
            "is_auto_generated": true
        }
    },
    "task_id": "task1"
}

**Output Schema** (Auto Mode Example):
{
    "is_automatic": true,
    "mode_used": "auto",
    "timestamp": 1712854012.123,
    "thoughts": {
        "technical": [
            "Your chart doesn't render because the 'data' field is empty. Try adding a 'values' object with your data.",
            "You may be missing 'encoding' definitions. Vega-Lite needs both 'x' and 'y' channels to position bars.",
            "Make sure you include a mark type and fields for encoding. A minimal bar chart requires: 'mark', 'data', 'encoding'."
        ],
        "heuristic": [
            "What fields are you trying to display on the x and y axes?",
            "How does your data structure match the encoding definition?",
            "Have you defined both the mark and the encoding channels required for this chart?"
        ]
    },
    "metadata": {
        "cognitive_level": "understand",
        "error_patterns": [
            { "type": "data", "message": "Missing values field" }
        ],
        "learning_history": {
            "preferred_mode": "technical",
            "completed_tasks_count": 4
        },
        "current_analysis": {
            "code": {
                "has_mark": true,
                "has_data": false,
                "has_encoding": false
            },
            "question": {
                "types": ["debug", "visualization"],
                "thinking_type": "A",
                "has_hypothesis": false
            },
            "is_automatic": true,
            "task_id": "task1"
        }
    }
}

**Output Example (Technical Mode Only)**:
{
    "is_automatic": false,
    "mode_used": "technical",
    "timestamp": 1712854012.345,
    "thoughts": {
        "technical": [
```

```
            "Try specifying your data like this:\n    \"data\": { \"values\": [ { \"x\": 1, \"y\": 2 }, { \"x\": 2, \"y\": 3 } ] }",
            "Your encoding might be missing. Add:\n    \"encoding\": { \"x\": { \"field\": \"x\", \"type\": \"quantitative\" }, \"y\": { \"field\": \"y\", \"type\": \"quantitative\" } }",
            "Also make sure your mark is set to 'bar'. Try this:\n    \"mark\": \"bar\""
        ]
    },
    "metadata": {
        "cognitive_level": "understand",
        "error_patterns": [
            { "type": "data", "message": "Missing values field" }
        ],
        "learning_history": {
            "preferred_mode": "technical",
            "completed_tasks_count": 4
        },
        "current_analysis": {
            "code": {
                "has_mark": true,
                "has_data": false,
                "has_encoding": false
            },
            "question": {
                "types": ["debug", "visualization"],
                "thinking_type": "A",
                "has_hypothesis": false
            },
            "is_automatic": false,
            "task_id": "task1"
        }
    }
}

**Behavior Rules**:

1. **Silent Mode**:
   - Do not generate any textual response.
   - Only return updated metadata (e.g., error pattern, cognitive level, etc.)

2. **Auto Mode**:
   - Generate both technical and heuristic responses.
   - Each response category must include 3 distinct, task-aligned responses.
   - Use historical error patterns and Bloom-level classification to customize difficulty.

3. **Technical Mode**:
   - Provide 3 fix-oriented, code-grounded responses.
   - Include complete but minimal code examples per response.
   - Ground each suggestion in task goals and data validity checks.

4. **Heuristic Mode**:
   - Provide 3 concise, question-led responses to stimulate student reflection.
   - Encourage self-correction and exploration without directly giving answers.
   - Align questions with the user's current cognitive level and task intent.

5. **Metadata Packaging**:
   - Always include structured metadata for downstream agent reasoning:
     * Bloom cognitive level
     * Error types and frequency
     * Learning history and style
     * Automatic vs. manual origin of message
     * Code validity and structure

**Technical Prompt Construction Example (in ThoughtFormationSystem)**

Depending on whether the message is automatically triggered or manually asked by the user,
the following templates are used to construct prompt for OpenAI API.

Case 1: Automatically Detected (is_automatic = true)

Prompt Template:
"""
As a proactive technical tutor, generate 3 different responses to automatically detected issues.
Each response MUST be completely separated from others using the '---RESPONSE---' marker.

Current Context:
- System Message: {user_message}
- Code: {current_code}
- Code Analysis: {code_analysis_dict}
- Question Analysis: {question_analysis_dict}
- Data Status: {"Valid" or "Invalid"}
{task_specific_context}

Historical Context:
- Cognitive Level: {cognitive_level}
- Error Patterns: {error_list}
- Learning process: {student_process}

For EACH response, please:
1. Proactively identify potential issues or improvements specific to the current task
2. Provide clear, actionable suggestions that align with the task's visualization goals
3. Include specific code examples that match the task requirements
4. Focus on best practices for the specific type of visualization needed
5. Keep explanations concise but comprehensive

IMPORTANT FORMAT REQUIREMENTS:
- DO NOT use any markdown formatting (no **, *, _, etc.)
- Use plain text only
- For emphasis, use UPPERCASE words instead of markdown
- For code examples, simply indent them with 4 spaces
- Keep line breaks for readability

Format your response exactly like this:

---RESPONSE---
[First proactive technical response in plain text]

---RESPONSE---
[Second proactive technical response in plain text]

---RESPONSE---
[Third proactive technical response in plain text]
"""

Case 2: User-Initiated (is_automatic = false)
```

```
Prompt Template:
"""
As a technical tutor, please generate 3 different technical responses to the user's question.
Each response MUST be completely separated from others using the '---RESPONSE---' marker.

Current Context:
- Question: {user_message}
- Code: {current_code}
- Code Analysis: {code_analysis_dict}
- Question Analysis: {question_analysis_dict}
- Data Status: {"Valid" or "Invalid"}
{task_specific_context}

Historical Context:
- Cognitive Level: {cognitive_level}
- Error Patterns: {error_list}
- Learning process: {student_process}

For EACH response, please:
1. Directly answer the user's question with explanation
2. Provide working code examples that align with the task's specific requirements
3. Build upon the student's historical learning
4. Explain why the suggested code works and is appropriate for this specific visualization task
5. Include complete, working code examples with necessary explanations, also need concise

Format your response exactly like this:

---RESPONSE---
[First technical response here with explanation and reasoning]

---RESPONSE---
[Second technical response here with explanation and reasoning]

---RESPONSE---
[Third technical response here with explanation and reasoning]
"""

"""
```

G Prompt for Selecting Adaptive Feedback Modes

```
    prompt = """
You are a teaching assistant AI responsible for selecting the most appropriate feedback style (technical or heuristic) and choosing the best response for the student.

You will receive:
1. Student's current question and code.
2. A set of generated technical and heuristic responses.
3. Analysis results about the student's cognitive level, past learning behavior, error patterns, and code structure.

Your task is to:
A. Determine whether the technical or heuristic feedback mode is more suitable for the current student situation.
B. Select the single best response from the chosen mode that maximally aligns with the student's needs.

Use the following decision framework:
- Mode selection is based on cognitive psychology and instructional strategy principles.
- Apply a weighted scheme to assess:
  * Current cognitive level (50%)
  * Error types (20%)
  * Learning history (30%)

You can refer to the principles:
- If the student is at the Apply level or above, with mostly logic/design errors and steady progress, prefer heuristic feedback.
- If the student is at lower cognitive levels, has frequent syntax errors, or shows inconsistent progress, prefer technical feedback.

Once a mode is selected, rank the candidate responses from that mode using the following five criteria:
1. Relevance to the student's question and code (40%)
2. Complexity appropriate to the cognitive level (20%)
3. Consistency with prior behavior and learning history (20%)
4. Clarity of explanation (15%)
5. Urgency based on current errors or stagnation (5%)

Choose the top-ranked response and justify your choice.

--- Input Example ---
student_question: Why is my bar chart blank?
student_code: { "mark": "bar", "encoding": {}, "data": {} }
thoughts: {
  "technical": ["Try adding a 'values' array to your data field.", "Ensure 'encoding' has 'x' and 'y' fields defined.", "The chart is empty because Vega-Lite can't draw bars without data values."],
  "heuristic": ["What is missing from the data definition?", "How does your encoding connect to the dataset?", "Can you see if you've declared both axes?"]
}
cognitive_info: {"level": "understand", "confidence": 0.7, "has_hypothesis": false}
error_info: {"patterns": [{"type": "data", "message": "Missing values field"}]}
learning_history: {"preferred_mode": "heuristic", "completed_tasks_count": 3, "success_rate": 0.6}
code_analysis: {"has_mark": true, "has_data": false, "has_encoding": false}

--- Output Format ---
{
  "selected_mode": "heuristic" or "technical",
  "selected_response": "...chosen response string from that mode...",
  "justification": "...explanation of why this response and mode were selected based on cognitive and code context."
}

--- Output Example ---
{
  "selected_mode": "heuristic",
  "selected_response": "What is missing from the data definition?",
```

```
  "justification": "The student's question is vague and the code lacks both data and encoding, which suggests early confusion. Their preferred mode is also heuristic, and their Bloom level is 'understand'. A reflective prompt would encourage self-discovery more effectively than a direct fix."
}

Please analyze carefully. Consider if the student is looking for a direct fix or needs guidance. If the question is vague or the code is incomplete, heuristic guidance may be better. If the error is obvious and the student has shown technical preference, technical feedback might be better.
"""
```

H Prompt for Intervening to Support Learning Progress

```
    prompt = """
You are a classroom intervention assistant AI responsible for deciding whether an instructor should intervene to support a student during programming activities.

You will receive:
1. The student's current feedback mode (e.g., auto, technical, heuristic)
2. Cognitive analysis (level, confidence, understanding)
3. Error analysis (types, frequency, severity)
4. Learning history (success rate, help frequency, completion rate)
5. Trigger event information (type: active, passive, predictive, and whether auto-generated)

Your task is to:
A. Decide whether intervention is necessary at this time.
B. Provide a justification for your decision based on cognitive need, error risk, history, and trigger information.
C. Assign an intervention score (0.0 to 1.0) and suggest an appropriate intervention mode ("proactive", "passive").

Use the following decision framework:
- Immediately intervene if explicit help is requested (passive, not auto-generated), or if the student is stagnant (active trigger with duration > 60s).
- Otherwise, compute a motivation-to-intervene score using:
  * Error severity and frequency (40%)
  * Cognitive level, confidence, and understanding (30%)
  * Historical performance (30%)
- If the combined score exceeds 0.5, initiate proactive intervention.
- If the score is below threshold, refrain from intervening to encourage autonomy and metacognitive development.

--- Input Example ---
current_mode: "auto"
cognitive_analysis: {
  "level": 2,
  "confidence": 0.4,
  "understanding": 0.3
}
error_analysis: {
  "patterns": [{"type": "syntax"}, {"type": "runtime"}],
  "frequency": {"syntax": 3, "runtime": 2}
}
learning_history: {
  "success_rate": 0.4,
  "completion_rate": 0.5,
  "help_frequency": 0.6
}
trigger_info: {
  "type": "active",
  "details": {"is_stagnant": true, "duration": 140}
}

--- Output Format ---
{
  "should_intervene": true or false,
  "intervention_score": float (0.0 - 1.0),
  "mode": "technical" | "heuristic" | "auto",
  "reason": "...rationale for the decision...",
  "timestamp": "YYYY-MM-DDTHH:MM:SS"
}

--- Output Example ---
{
  "should_intervene": true,
  "intervention_score": 0.85,
  "mode": "proactive",
  "reason": "Student shows low confidence and understanding, has high help frequency, and experienced stagnation for 140 seconds. Proactive intervention recommended to re-engage the student and provide timely support.",
  "timestamp": "2025-04-11T22:45:00"
}

Please weigh all inputs thoughtfully and act in the student's best interest.
"""
```

I Performance Comparison of Different LLMs

To clarify the engineering considerations underlying our choice of the large language model (LLM), we conducted a controlled comparison of multiple GPT models using five representative query types derived from classroom use. These query types include basic visualization construction, error diagnosis and repair, feature enhancement, conceptual explanation, and task oriented specification generation. It is important to emphasize that this comparison focuses on system level metrics rather than semantic correctness or task optimality. Specifically, we evaluated average response latency, token usage, and output format compliance. Format compliance measures whether model outputs strictly adhere to the response schema required by *ClassAid*, such as correct delimiters, complete response blocks, and parsable code. This property is critical for enabling instructor mediated orchestration and downstream processing in real time classroom settings.

```
[
  {
    "query_type": "Q1: Basic Visualization Construction",
    "user_message": "How do I create a bar chart in Vega-Lite?",
    "current_code": "",
    "description": "Constructing a basic bar chart specification from scratch"
  },
  {
    "query_type": "Q2: Error Diagnosis and Repair",
```

```
    "user_message": "My code has an error: Missing
     encoding specification",
    "current_code": {
      "$schema": "https://vega.github.io/schema/vega-lite
     /v5.json",
      "data": {"values": [{"x": 1, "y": 2}]},
      "mark": "bar"
    },
    "description": "Identifying and repairing missing
     encoding fields"
  },
  {
    "query_type": "Q3: Feature Enhancement",
    "user_message": "How can I add colors to distinguish
     different categories?",
    "current_code": {
      "$schema": "https://vega.github.io/schema/vega-lite
     /v5.json",
      "data": {"values": [{"category": "A", "value": 10},
      {"category": "B", "value": 20}]},
      "mark": "bar",
      "encoding": {
        "x": {"field": "category", "type": "nominal"},
        "y": {"field": "value", "type": "quantitative"}
      }
    },
    "description": "Extending an existing visualization
     with color encoding"
  },
  {
    "query_type": "Q4: Conceptual Explanation",
    "user_message": "What's the difference between bar
     and column charts?",
    "current_code": "",
    "description": "Explaining visualization concepts
     without code generation"
  },
  {
    "query_type": "Q5: Task-Oriented Specification
     Generation",
    "user_message": "I want to create a histogram showing
      score distribution",
    "current_code": {
      "$schema": "https://vega.github.io/schema/vega-lite
     /v5.json",
      "data": {"values": [{"score": 85}, {"score": 90},
     {"score": 75}]}
    },
    "description": "Generating a complete visualization
     specification based on task intent"
  }
]
```

As summarized in Table 10, all evaluated models were compared based on the average results from 30 experimental runs. While all the models were generally capable of generating valid responses, their performance varied significantly across different query types. Other models, such as GPT 4 and GPT 4 Turbo, tended to generate longer and more detailed explanations but exhibited significantly higher response latency, particularly for complex task oriented queries. This characteristic makes them less suitable for real time classroom deployment. In contrast, GPT 4o consistently demonstrated a more balanced trade off among response latency, instructional richness, format compliance, and per query cost.

Based on these engineering considerations, we adopted GPT 4o for the classroom deployment of *ClassAid*. We emphasize that our contribution does not depend on any specific LLM, but rather on the system design that enables real time instructor control over student AI interaction. The appendix level model comparison is intended to improve transparency and support future extensions of this work under alternative model choices.

Table 10: Performance and Cost Comparison of LLMs Across Query Types (Q1–Q5)

Model	Metric	Q1	Q2	Q3	Q4	Q5
GPT-4o	Avg. Response Time (s)	7.33	10.51	9.00	5.51	10.72
	Avg. Tokens / Request	563	966	1096	512	1101
	Format Compliance Rate (%)	100	100	90	100	100
	Input Cost ($/1M tokens)			5.00		
	Output Cost ($/1M tokens)			15.00		
GPT-4-turbo	Avg. Response Time (s)	12.50	14.40	17.81	12.95	19.01
	Avg. Tokens / Request	698	989	1129	651	1142
	Format Compliance Rate (%)	100	100	100	100	53
	Input Cost ($/1M tokens)			10.00		
	Output Cost ($/1M tokens)			30.00		
GPT-4-0125-preview	Avg. Response Time (s)	10.71	19.94	41.48	11.65	19.86
	Avg. Tokens / Request	602	1119	1214	662	1170
	Format Compliance Rate (%)	100	93	100	100	83
	Input Cost ($/1M tokens)			10.00		
	Output Cost ($/1M tokens)			30.00		