ASPIRE: Identifying Real-Time Behavior-Level Activity Signals of Student Coding Understanding with LLMs

Anonymous Author(s)

ABSTRACT

Programming instructors often rely on observable signals such as students' code issues or proactive help seeking behaviors to determine how to allocate their support resources. These signals, however, may overlook the nuanced and multifaceted nature of individual learning processes. To direct instructors' attention towards students who may need support but are not identified using traditional metrics, we introduce ASPIRE. ASPIRE uses Large Language Models (LLMs) to provide instructors with additional signals about students' understanding while completing an in-class coding exercise. These signals are derived from behavior-level activities, such as feedback integration behavior, and are then visualized using a hierarchical structure to help streamline instructors' efforts to promptly allocate support where it is most needed. A within-subject study with 12 instructors demonstrated that ASPIRE facilitated instructors' identification of novel learning behaviors and that these behaviors enhanced their baseline insights.

ACM Reference Format:
Anonymous Author(s). 2024. ASPIRE: Identifying Real-Time Behavior-Level Activity Signals of Student Coding Understanding with LLMs. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 11 pages. https://doi.org/

1 INTRODUCTION

In-class coding exercises are common ways for programming instructors to provide students with practical coding experience. The timely identification of students' misconceptions during these exercises is critical to instructors' offering prompt support to alleviate learning obstacles. Existing practices to identify students' struggles, however, rely on standard measures, test case pass rates, or the questions students ask, which are not scalable to large class sizes nor do they align with students' actual understanding or their need for support (e.g., distinguishing between a student who is struggling from one who moves at a slower pace but still meets success criteria). Assessing students using limited data points, such as a 1-minute snapshot of an exercise where a student has a 20% pass rate, may not be an accurate signal of which students need help as these data points do not capture important nuances in individual coding styles or progress.

Past systems such as Codeopticon [11] allowed instructors to view students' code in real-time, but they required instructors to read code individually, making them unscalable to large classes. Visualizations such as VizProg [31] provided overviews of student progress, but focused on a single level of representation, thus losing important structural and semantic cues. Other large-scale code analysis interfaces such as Overcode[10] overlooked many coding process level behaviors (e.g., students who complete tests perfectly and quickly versus those who are slower and more methodical). To effectively support all types of learners, there is a need for systems that help instructors disambiguate between individual variations in coding process style and the new challenges students face.

Recent progress in Large Language Models (LLMs) has enabled for the identification of important facets of language and support for real-time interaction [3]. For instance, LLMs have acted as support agents during TA training [18], provided personalized tutoring [13], helped raise awareness about alternative aspects (e.g., role playing) within a group of people [6], and developed empathetic responses that adapt to users' emotions [1]. Leveraging these advances, we propose ASPIRE, a system that uses LLMs to identify students' intrinsic coding competencies beyond simple pass rates and code issues so instructors can allocate support to those who need it most. ASPIRE consists of a student UI and an instructor UI. In the student UI, each student interacts with a chat-based AI learning assistant that offers context-aware feedback and support as they complete coding tasks. The AI chat bot captures nuances in students' behaviors, such as typing patterns and their ability to integrate the assistant's responses effectively, forming a model of each student's performance at regular intervals. These models are then aggregated and visualized in the instructor UI, which enables instructors to identify students who may require additional support. Furthermore, instructors can review the AI's decision-making criteria by examining the explanations and evidence that are provided.

To evaluate how effective ASPIRE can be at helping participants identify students with coding issues, we conducted a within-subject study that asked participants to use ASPIRE and a baseline system. Study participants, all of whom had prior teaching experience, reported that ASPIRE lowered their effort when identifying struggling students because it provided an experience similar to receiving well-structured updates about students' performance from a teaching assistant. Moreover, the associated explanations largely aligned with the participants' own intuitions. In cases where ASPIRE differed from participants' intuitions, they found that ASPIRE provided a different, yet insightful, viewpoint. Our findings suggest that ASPIRE has the potential to greatly enhance the effectiveness of in-class programming exercises by providing instructors with a more efficient means of understanding and supporting large groups of students. Thus, our work provides three contributions:

- A demonstration of how LLMs can be used to scalably understand students' coding performance in real-time during in-class programming exercises.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/

- ASPIRE, an interactive system that presents structured visualizations of student coding performance, enabling instructors to easily identify students who may require additional support and explore the AI's reasoning through associated explanations and evidence.
- Empirical evidence demonstrating that ASPIRE lowers instructors' effort when identifying struggling students and provides complementary insights, even when the AI's perspective differs from the instructor's own intuition.

2 RELATED WORK

Our work builds upon prior research within three areas: learning programming at scale, LLMs in CS education, and code visualization at scale.

2.1 Learning Programming at Scale

As programming courses grow in size, it becomes increasingly challenging for instructors to monitor and support students' learning progress during in-class coding exercises due to time and spatial constraints [17, 28]. Additionally, prior research has found that early-stage students often have difficulty expressing their questions and assumptions clearly [5, 31]. Existing tools to monitor student progress have limitations when it comes to scalably identifying student mistakes in real-time. Codeopticon [11], for example, enabled instructors to view students' coding process and chat with them individually, but this approach is scalable to large classes. Visualizations such as VizProg [31] provided overviews of students' progress, but focused on a single abstraction level, losing important structural and semantic cues. Other tools, such as Nbgrader [4], Overcode [10], and Autostyle [19], generated asynchronous feedback at scale but were not applied to real-time feedback generation. Together, it remains unclear how to support instructors to easily and effectively assess students' performance at scale.

2.2 LLMs in Computer Science Education

Recent advancements in LLMs have shown promising results in various educational applications, particularly in the field of computer science education [29]. In general, AI has the potential to scale and enhance instructors capabilities by immediately producing helpful responses to frequently asked questions [13], selecting appropriate practice problems for students [23], and enabling instructors to create course-specific intelligent tutoring systems [16]. By leveraging LLMs, instructors can better understand how many and which students are struggling, the problem-solving approaches that students use, and the speed of student progress. This information can enable instructors to adapt their in-class exercises to meet current student needs, such as deciding when to help individual students, extending the time given for an exercise, addressing common issues with the whole class, or grouping students into mixed teams for group exercises.

Understanding AI-generated output, however, is challenging due to potential biases and inaccuracies in the content, which may lead instructors to make decisions that unfairly impact students. Prior work within explainable AI (XAI) has explored visualization techniques to support the sensemaking of LLM output, such as transforming output into diagrams [8, 14, 26]. These approaches represented the connections between concepts in a graphical format to assist users in understanding and evaluating LLM responses, however, they were not designed to handle the scale and real-time requirements inherent in live learning contexts. Furthermore, they prioritized the organization of information rather than addressing biases and verifying the accuracy of output.

Another growing area of research within XAI has been how to effectively prompt users to verify the correctness of AI-based output. Prior work has shown that when done well, AI explanations can help achieve human-AI team performance that surpasses humans or AI alone [2]. However, most explanations such as displaying AI accuracy or confidence, haven't exceeded baseline methods, suggesting that explanations can increase overreliance, which is troublesome when an AI makes errors. Recent work by Fok and Weld argued that explanations are only useful to the extent that they allow a human decision maker to verify the correctness of the AI's prediction [7]. Inspired by this, we aim to help instructors to quickly interpret and verify the AI's output to ensure its accuracy and fairness during in-class coding exericses.

2.3 Code Visualization at Scale

Visualizing student learning behaviors, such as help-seeking patterns and struggles, in real-time is challenging due to the multi-faceted nature of student behavior data (e.g., code submissions, interactions with learning resources, and social interactions). To effectively support instructors in monitoring and understanding student progress, it is necessary to redesign visualizations using structured representations that align with instructors' mental models and informational needs.

Prior research has explored various approaches to cluster student code submissions to reduce the number of variations instructors need to handle manually [30]. These approaches include using Abstract Syntax Tree (AST) edit distances to evaluate syntax and functional similarity [22], analyzing data submissions of programming exercises by the solution strategies that were used [15], and determining semantically equivalent code snippets to efficiently index MOOC programming assignments [21]. Tools like Foobar [9], Overcode [10], and MistakeBrowser [12] used static and dynamic analysis to cluster similar code submissions and provided visualizations to help instructors understand code solution variations and students' bugs. Piech et al. introduced a method to encode student programs as embeddings in neural networks and supported feedback generation at scale via the clusters learned in the embedding space [22].

In addition to clustering tools, there have also been tools that supported the comparison between programs, such as Microsoft Win Diff, which highlighted text differences between files and MOS to detect plagiarism among student programs [24]. Taherkhani et al. used a code embedding method to cluster sorting algorithm implementations [27]. Within the growing field of sensemaking for LLM output, it has become essential to consider how to effectively visualize AI's responses and explanations in the context of education [8]. Instructors need to be able to quickly interpret and verify an AI's justification for its models of student performance to ensure fairness and accuracy.

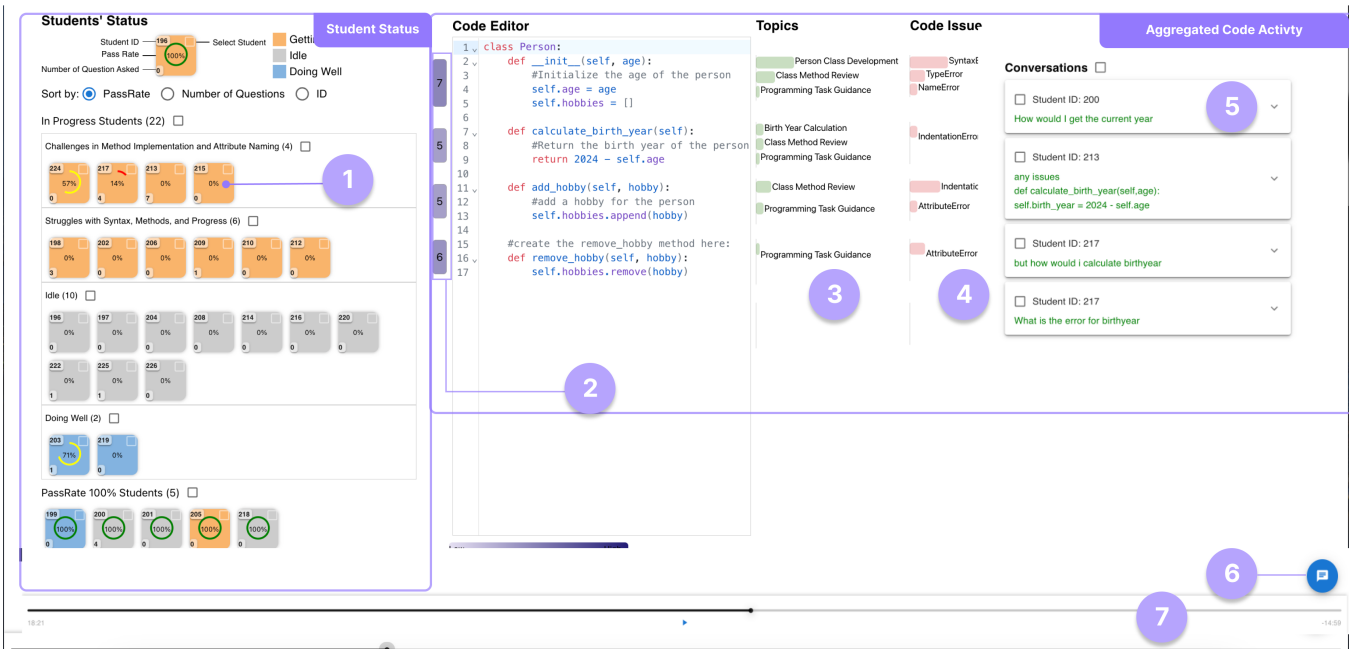

Figure 1: ASPIRE's Instructor User Interface, which included three panels: (1) the AI-summarized student performance panel, (2) a code editor panel with a heatmap depicting the edit frequency of students, (3,4) and an aggregated topics and code issue panel with a bar chart of the current count of conversation topics and code errors. (5) A list of details about each data point in the bar chart. (6) Instructors can provide feedback to students based on their performance. (7) Instructors can control the playback of the session using the timeline.

3 ASPIRE SYSTEM

Motivated by insights derived during prior work, ASPIRE was designed to address the challenges instructors face while understanding students' performance and provide timely support during large programming courses. Three design goals guided the development of ASPIRE:

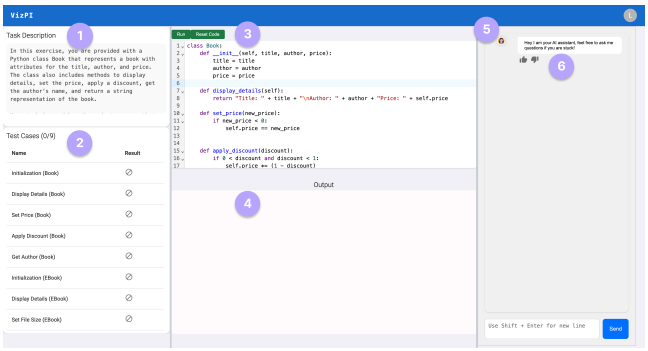

Figure 2: ASPIRE's Student User Interface, which included five panels: (1) a task description panel, (2) a unit test panel, (3) a code editor to run code or reset to default code, (4) a code output panel, and (5) and a chat interface that contained an AI chat bot. In this interface, (6) students could rate the quality of feedback from the AI chat bot.

3.1 Design Goals

- **DG1:** Enable instructors to easily and effectively understand student performance to help them accurately observe student progress and provide timely support in large programming courses.
- **DG2:** Ensure instructors can quickly interpret and verify AI models of student performance to ensure their fairness and accuracy.
- **DG3:** Present AI-generated explanations to bridge the gap between the AI's insights and instructors' understanding to enable instructors to make informed decisions and provide timely support to students.

ASPIRE's dashboard was divided into two user interfaces, i.e., one for students and one for instructors (Figure 2).

3.2 Student UI

The Student UI features a code editor and an AI chat bot. While students write their solution to small Python exercises in-class, they can refer to the task instructions (Figure 2.1) and modify seed code (Figure 2.3). They can also obtain feedback and check the accuracy of their code by referring to the Code Output Panel (Figure 2.4) and an autograder (Figure 2.2). The AI chat bot provides exercise-specific feedback when a student seeks help using information such as the current state of the code, the task description, the pass rate status, and previous conversations the student has had (Figure 2.5). The AI will not generate code for students but rather provide

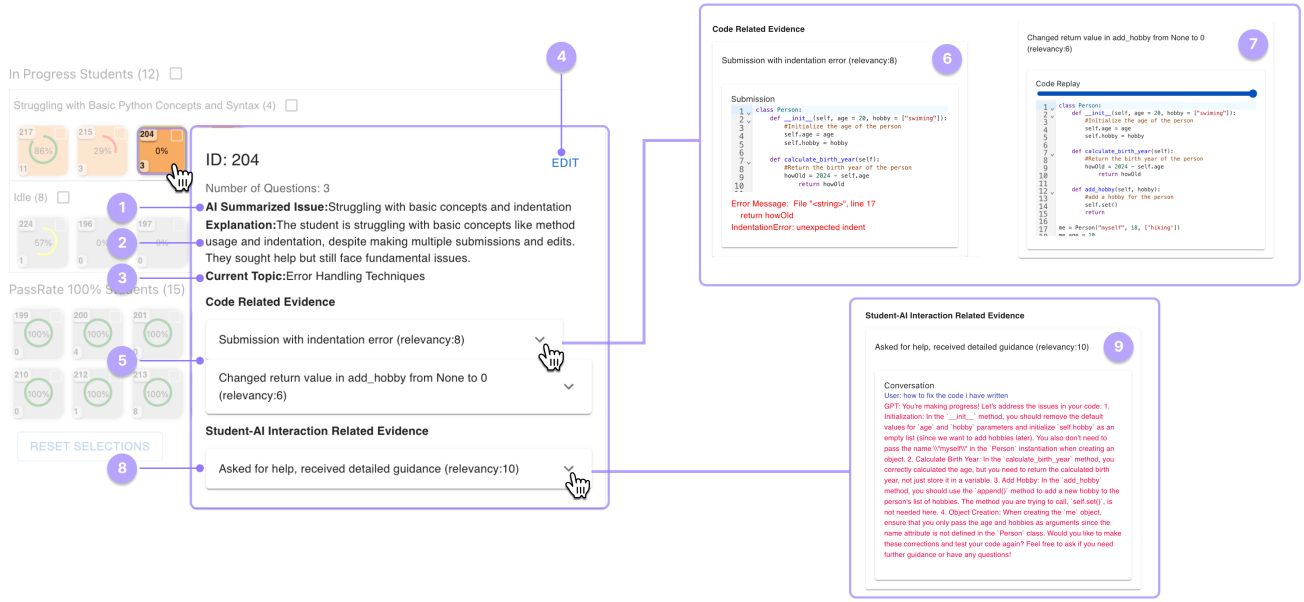

Figure 3: Hovering on a Status Tile within ASPIRE revealed a pop up containing (1) an AI-summarized issue, (2) an explanation, and (3) the current topic. Instructors could edit these outputs using (4) the edit button. ASPIRE also showed code-related evidence about (5) the current issue. Clicking on the drop-down menu revealed code-related evidence, such as (6) an error message or (7) a student's edits over time. ASPIRE also summarized (8) student-AI interactions (i.e., conversations) about the current issue. (9) Clicking on the associated drop-down menu revealed the related conversation.

them with clues about how to finish their task, similar to how TAs and professors help students when they are stuck on assignments. Students can also provide feedback about the response from the AI chat bot (e.g., "Like" or "Dislike"; Figure 2.6).

3.3 Instructor UI

The Instructor UI presents a real-time dashboard of student performance using two panels, (i.e., Student Status, Aggregated Code Activity). Central to each panel is the *Status Tile*, a novel mechanism that ASPIRE uses to provide a visualization of student performance.

3.3.1 Status Tile(Figure 3). Status Tiles provide instructors with a summary and detailed analysis of student performance to help instructors understand current student performance [DG1]. The attributes that are displayed in a Status Tile include (Figure 4.1):

(1) **ID:** The unique identifier assigned to the student.
(2) **Pass Rate:** The percentage of unit tests that the student's code submission passed.
(3) **Number of Questions Asked:** The number of students asking a question to the AI chat bot
(4) **Current Status:** The performance of a student, which was based on their code editor activity and their programming task completeness which was categorized as:
   (a) *Idle:* The student is not actively engaged in coding or making progress.
   (b) *Getting Stuck:* The student is struggling with syntax and concepts and is not seeking or effectively using the AI assistance, thus leading to consistently low success on their coding exercises.
   (c) *Doing Well:* The student is engaged, understanding coding concepts, effectively using the AI assistance, and is showing consistently high or improving success on their coding exercises.

Hovering over a Status Tile will display more information about the difficulty the student is encountering [DG2]. For students classified as *Getting Stuck* or *Idle*, ASPIRE summarizes how students are struggling, thus enabling instructors to inspect students who are struggling without having to search through each student's conversation and code editor history. ASPIRE provides a one line summary of student performance (e.g., *Struggling with syntax and class structure*) (Figure 3.1) and a detailed explanation of why ASPIRE concludes the student is having difficulty (e.g., *The student is actively editing their code but consistently fails to correct syntax errors, thus indicating a struggle with basic Python syntax, particularly in method definitions and class structure*)(Figure 3.2) [DG3]. Such summaries outline the reason why the students are having difficulty based on the analysis of the code activities and conversations with the AI chat bot. Instructors can edit the generated summaries if they find them to be inconsistent with the student's actual performance (Figure 3.6). This allows instructors to verify the accuracy of AI models of student performance. If the instructor edits the generated summaries, ASPIRE will adjust its existing summaries based on the instructor feedback.

For each student, the code activity related to the summarized issue is displayed as a drop-down list (Figure 3.4) [DG3]. Each item

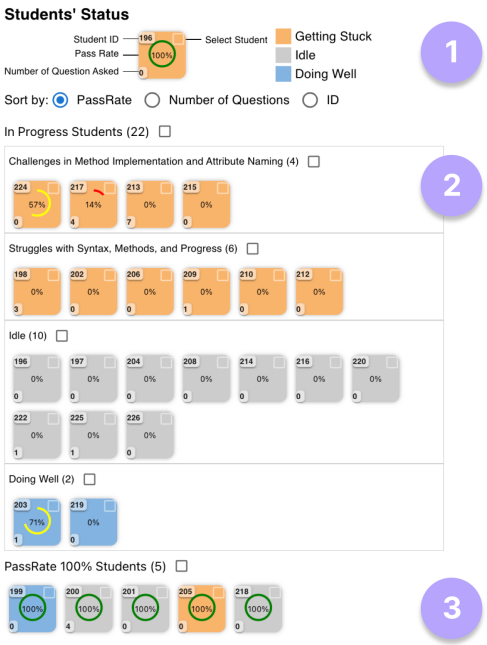

Figure 4: The AI-summarized Student Performance panel that had (1) a legend of the Status Tiles, (2) a summary of in-progress students grouped by student performance, and (3) a grouping of all students who passed all unit tests.

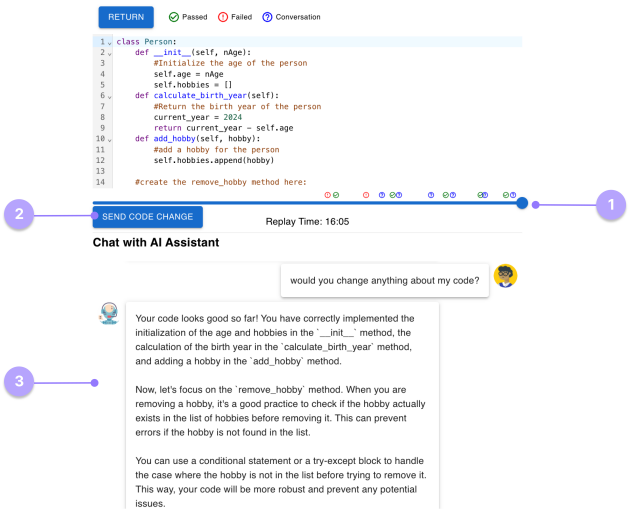

Figure 5: Activity Details Panel.(1) Timeline that shows student's passed submission and conversation with AI chat bot. (2) Instructors can modify student's code and give feedback. (3) Conversation history of the student and the AI chat bot.

in the drop-down list includes a heading summarizing the code activity and its relevance to the issue described on the Status Tile. Two types of code activities are documented in this panel, *code errors* (i.e., errors thrown by the students' code submission that are relevant to the summarized issue) (Figure 3.7) and *editing history* (i.e., a summary of student code editor activity that hints at the summarized issue) (Figure 3.8). Expanding a code error displays the code submission that contains the specified errors, while expanding the editing history shows a playback of the edits that the student made.

ASPIRE also documents evidence of a student's interactions with the AI chat bot whenever they are struggling [DG3]. The conversation topic between the student and AI chat bot is summarised (Figure 3.3), and the chat message evidence is organized using a drop-down menu with a summary of the conversation as the heading (Figure 3.5). When expanded, instructors can examine conversation excerpts between the student and the AI chat bot (Figure 3.9).

3.3.2 Student Status Panel. This panel showcases the AI's perspective or model of students' performance to provide instructors with an overview of how students are progressing through the exercises and identifies those who may require additional support [DG2]. Based on the AI-summarized issue displayed on the Status Tile, ASPIRE first classifies students as In-Progress or Pass Rate 100% (Figure 4.2 , 4.3). Then, based on the AI summarized issue in the Status Tile, students in the In-Progress group with similar struggles are grouped together (Figure 4.2). For each group, instructors can sort each Status Tile by id, pass rate, number of questions asked, or their current status.

3.3.3 Activity Details Panel. Clicking on a Status Tile opens the Activity Details panel, which features a code editor, a playback slider, and a chat interface (Figure 5). This panel enables instructors to delve into the specifics of each student's activities, offering insights into student's coding progress, interactions with the AI chat bot, and areas of potential difficulty [DG1]. This enables instructors to provide precise support and guidance where it's needed most. The code editor displays the code implemented by the selected student. By moving the slider, instructors can observe the evolution of the student's code over time (Figure 5.1). The playback slider is annotated with three symbols to indicate different student activities, i.e., an exclamation point for submissions that failed some unit tests, a question mark for instances where the student sought assistance from an AI chat bot, and a check mark for submissions that successfully passed all unit tests. These symbols provide a chronological perspective about students' submissions and their requests for help, enhancing an instructor's understanding of their learning. Instructors can edit the student's code directly from this panel or propose changes (Figure 5.2). Additionally, instructors can playback the student's conversation with the AI chat bot to gain a deeper understanding of how the student utilized the AI chat bot to help them to debug and problem solve (Figure 5.3). Chat messages are shown by controlling the playback.

3.3.4 Aggregated Code Activity Panel. This panel presents a summarized view of students' coding activities, such as the frequency of code edits for different functions, the code errors within each submission, and the usage of the AI chat bot [DG1]. This information

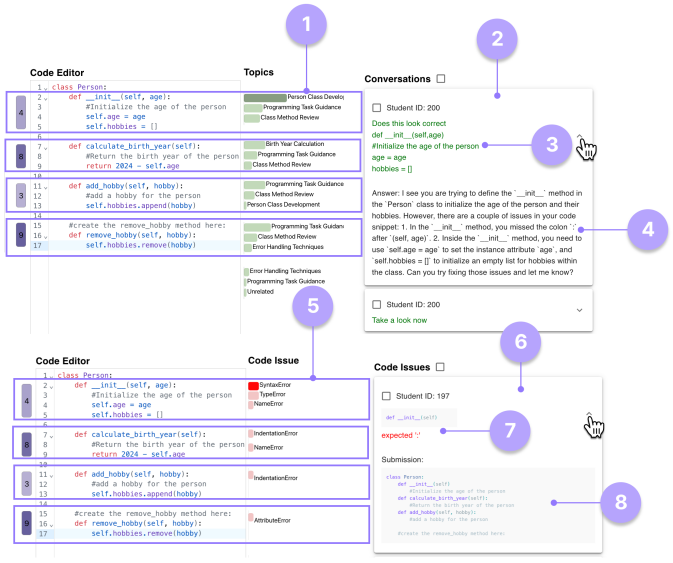

Figure 6: Aggregated Code Activity Panel. (1) Bar charts showed an aggregated count of conversation topics. (2) Clicking on the bar (highlighted in dark green) revealed details about the data points, which shows (3) the lines of code related to the topic and (4) the related conversation with the LLM. (5) Similarly, clicking on a bar for the aggregated count of code issue (highlighted in dark red) (6) revealed details about the selected code issue, which contains (7) the error message and (8) the associated code submission.

helps instructors gauge the overall progress and engagement of the class. This panel is divided into a code editor that has a heatmap, an aggregated topic bar chart, and an aggregated code error bar chart. Each element is aligned in the same row with the corresponding function for easier data interpretation (Figure 6.1)

Inspired by Porta [20], which uses profiling visualizations for tracked user activity, we created a heatmap to highlight edit frequencies across students (Figure 1.2). The heatmap shows the number of students editing each function, with the brightness of the color correlating to the number of edits for each function. Each edit is defined by an edit selection on the corresponding function (i.e., a mouse click on the lines of code of the corresponding function). The frequency of each click was normalized so it mapped to a value between 0 and 1, with 0 being no student activity on the corresponding function and 1 being all students having interacted with the corresponding function. Then, the resulting color value was interpolated based on the normalized value and displayed underneath the number of students editing the code on the heatmap. The color change of the heatmap across time acts as a proxy of how many students are stuck on a particular section of code. The topic and code issue bar charts display the conversation topics and the code issues found for each function (Figure 1.3, 1.4). The bars in each chart are sorted in descending order by the occurrence of the topic or errors.

- **Topics: (Figure 6.1)** The topics displayed in the bar chart summarize the type of conversations had by each group with the AI chat bot at a given time. Clicking on a bar reveals details about all the students that had on the engaged conversation topics with the AI chat bot (Figure 5.2). Detailed conversation topics are presented as collapsible cards that show the chat message posed by the student (Figure 5.3). The drop-down menu shows the response generated by the AI chat bot in response to the question asked (Figure 5.4).
- **Code Issues: (Figure 6.5)** Code issues are the compiler errors students encountered while completing their programming task. The bar chart depicts an aggregated list of the compiling errors that occurred when students ran the corresponding function. Similar to topics, clicking on the bars will display code issue details, which are organized as collapsible cards (Figure 6.6) that display the line an error was thrown and the associated error message (Figure 6.7). The drop-down menu displays the student implementation of all the functions, enabling an instructor to inspect the entirety of a student's code (Figure 6.8).

4 USER STUDY

To evaluate the effects of AI assistance on instructors' abilities to understand student coding performance in real-time, we conducted a lab-based user study using a within-subjects design. Our study was reviewed and approved by the Internal Review Board (IRB) at our university.

4.1 Participants

For this study, we recruited participants who were programming language instructors that monitored and worked with students' coding submissions at a large scale. Our recruitment efforts spanned multiple university mailing lists and Slack channels. Within these participant pools, we recruited 12 individuals (i.e., 9 Male, 3 Female; average of 6 years Python programming experience) who were 18 years or older, had teaching experience (e.g., instructor, TA), and had experience viewing students' behaviors and Python submissions at scale. Each participant was compensated with a $25 gift card in recognition for their effort and time.

4.2 Experimental Conditions and Data

As tools similar to ASPIRE do not exist, we create a baseline version of ASPIRE to use during our study. In this version, we removed all the AI generated information except for the topic summarization in bar chart. In the left panel, each Status Tile could only be assigned one of two statuses: Active or Idle. These statuses were assigned based on the students' activity over the previous minute (e.g., if they had taken no action, they were assigned Idle; otherwise they were assigned Active). The right panel and Individual View were the same as ASPIRE.

To ensure the authenticity and validity of the data used in the study, we utilized data that was collected during a Python programming course at our institution. Our data was collected by ASPIRE Student's platform with the help of 2 instructors. During the data collection process, we did not obtain any personal information about students. Instead, we collected their keystroke-level editing

data, the code they ran, error messages they encountered, their pass rates, and their conversations with the AI chat bot. The class exercises were voluntary, and students participated willingly. These exercises were not used as a basis for any grading during the course.

4.3 Procedure and Tasks

The study had three main phases: an instruction phase, a task phase, and an exit interview phase. During the instruction phase, participants were asked to complete a pre-study survey about their demographics. Then, the experimenter explained the research goals of the study and explained the tasks participants would perform.

For the task phase, participants completed two quizzes, one using the baseline system and one usingASPIRE. Each quiz consisted of multiple-choice and open-ended questions. The order of the two conditions was randomized. To minimize potential learning effect caused by using the same dataset and ensure that the overall complexity of the tasks remained consistent, the timestamps used during each task was different. At the beginning of each condition, the experimenter introduced the features and functionality of the condition system and participants were then instructed to use the system. After each quiz, participants were asked to complete a re-defined NASA-TLX questionnaire.

After the quizzes were completed, the exit interview phase began. During this phase, open-ended semi-structured questions were used to gather qualitative feedback about the visualizations, participants' mental models, their user experiences (e.g., *"Which feature do you think that can help you understand students' mental model effectively and why?"*), trustworthiness with the two systems (e.g., *"What do you think about the AI summarized performance? Were you able to trust it?"*), and potential improvements to ASPIRE.

4.4 Tasks

To evaluate the ASPIRE's ability to assist instructors in identifying struggling students, the first and second task asked participants to pinpoint struggling students at a particular timestamp, with additional requirements such as having sought help from an AI or being active within the past minute. For the third task, we aimed to assess the system's proficiency in aiding instructors to recognize common behavior patterns. Thus, participants were asked to describe the behavior patterns and common issues exhibited by struggling students at a given timestamp.

A fourth task was performed during the ASPIRE condition to evaluate the impact of using the edit feature on the user experience. After participants completed the first three tasks and the NASA-TLX survey in ASPIRE condition, they were asked to find one student in a subcategory of the In-Progress students whose AI generated summary was inconsistent with participant's observation and modify the AI generated summary. We then updated the data for all students after they submitted their modification. We then asked them to talk about the experience on being able to overwrite the AI's result and becoming involved in the process.

4.5 Data Analysis

Several measures were computed from our collected data. First, we analyzed the screen recording of each participants, and calculated the average time spent on each task1. We also analyzed participants'

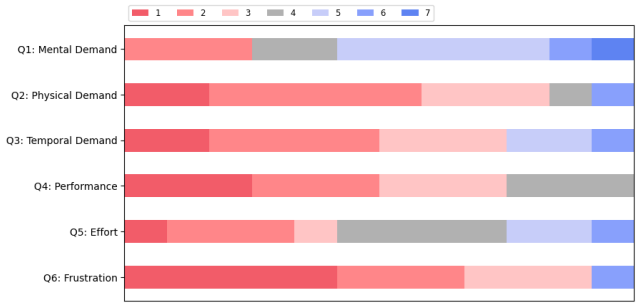

Figure 7: NASA-TLX results for the for the ASPIRE condition.

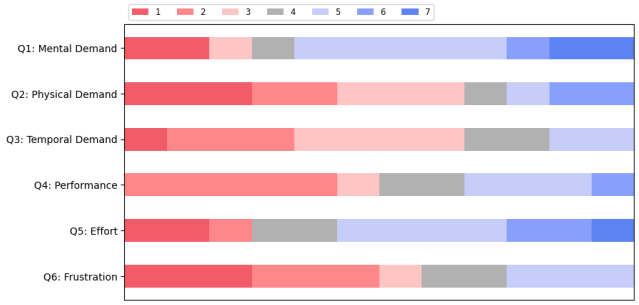

Figure 8: NASA-TLX results for the baseline condition.

responses to each multiple-choice question for each quiz. A member of the research team compiled a list of correct answers to the quiz questions by reviewing the replay. The quiz accuracy was calculated as:

$$\frac{\mathrm{M}}{\max(\mathrm{C}, \mathrm{S})}$$

where,

- M is the Number of Matched Answers, indicating how many selected answers match the correct answers.
- C is the Total Number of Correct Answers.
- S is the Total Number of Selected Answers.

Lastly, the authors also transcribed the interview audio recordings and performed a thematic analysis that led to XXX codes [25]. The authors conducted axial coding by revising, merging, and removing the initial codes, and identifying emerging topics.

5 RESULTS AND DISCUSSION

5.1 Time on Task

We noticed that for the first two tasks, participants were slower when using ASPIRE (i.e., 3.21 minutes and 2.79 minutes), compared to using the baseline system (i.e., 2.92 minutes and 2.25 minutes). For the third task, participants were faster while using ASPIRE (i.e., 2.92 minutes) than while using the baseline (i.e., 4 minutes).

	Task 1		Task 2		Task 3	
	M (min)	SD	M (min)	SD	M (min)	SD
Baseline	2.92	1.76	2.25	0.99	4.00	2.92
System	3.21	3.06	2.79	1.59	2.92	1.92

Table 1: Statistical results about average completion time with standard deviation spent on tasks for identifying behavior-level activities.

5.2 Accuracy

During the first and second tasks, participants were more accurate when using the baseline system (i.e., Task 1 = 0.56, Task 2 = 0.71) then when using ASPIRE (i.e., Task 1 = 0.45, Task 2 = 0.59).

	Baseline		System	
	Task 1	Task 2	Task 1	Task 2
Precision (M)	0.94	0.93	0.79	0.86
(SD)	0.15	0.30	0.11	0.37
Recall (M)	0.55	0.65	0.77	0.85
(SD)	0.15	0.37	0.19	0.31

Table 2: Statistical results for average values and standard deviation of precision and recall for completing tasks.

5.3 Post-Study Questionnaire

Based on the mean and median values, there was no significant difference between ASPIRE and baseline (Figures 7 and 8).

5.4 Qualitative Feedback and Discussions

Our thematic analysis uncovered several interesting insights into the use of systems such as ASPIRE.

5.4.1 ***AI Summaries Facilitate the Identification of Activity Signals.*** In general, the Student Status panel (Figure 1.1) enabled participants to identify behavioral activity signals within coding exercises. All participants noted that the Student Status panel, which was integrated using a hovering window (Figure 3), was informative and granular. These designs presented rich details about students' behavioral-level activities in minutes. Participants discerned that the Student Status panel was effective in recognizing and promptly allocating support for certain students.

> *"The topic part is definitely give me lots of, uh, insight about. Uh, what kind of, challenges they meet, based on the size of the plot and based on descriptions. Like this student's struggles stuck for a long time, but, he really try his best to ask the problem but still no progress. So that means probably at that time I can ask for to give him some support."[P4]*

Participants also appeared to change their strategy to find struggling students. During baseline condition, all participants dove into individual student details, check their history edits, submissions, and conversation history one by one, and tried to analyze their code issue by carefully looking at their code, progress, and the content of student-AI conversations. During the ASPIRE condition, several different behavioral patterns were observed. Eight participants only looked at the "Getting Stuck" students, with 4 of these participants quickly checking several AI summaries before providing their answers. three participants first looked at the sub-titles within the "In-Progress Students" section, and randomly sampled some summaries in each group. Then they dove into the details to quickly check if the AI's perspective aligned with their observations. If they thought AI's perspective was consistent with their own observation, then they tended to accept the AI's decision. As P0 mentioned:

> *"I think it's pretty, oh, helpful. I don't have to... maybe I would check one of them [students' details] to see if that's the case, but I definitely not have to check all of the students. So maybe this [AI summary] will be more helpful because it at least it has some summary for me." [P0]*

While using the ASPIRE for identifying real-time behavior-level activity signals of coding exercises, ASPIRE allowed instructors to better identity struggling students with weak or even no observable signals compared to the baseline condition. Instructors could detect students who were having similar problems at the moment in batch based on the AI-generalized information. However, in the baseline, without high-level information, participants usually dived into individual submission histories to understand their mental models and decide whether students are looking for additional help.

Among the 12 participants, only one looked into each student's details, checked them all, and made the decision by themselves. This participant considered the information contained within the Student Status panel to require high mental demands and codes that were more straightforward than textual descriptions.

5.4.2 ***Formation of Holistic Views of Common Student Behaviors.*** The responses to the open-ended questions revealed that, during the baseline condition, 8 out of 12 participants relied on the bar chart and heat map to identify top topics and code issues. Additionally, 8 participants reviewed all students' history codes and conversations to confirm their answers. During the ASPIRE condition, 9 of 12 participants reviewed the AI summary by checking the categories' titles for overview insights or reading the summarized issues and explanations as the first step. Instead of checking students' submissions individually, participants were able to provide answers with more descriptive observations by going through one or two students within each AI-generated cluster.

Participants also mentioned that AI summary provided them with a "high-level understanding" of all students at a glance. Compared to identifying bar charts or checking student submissions separately, these participants felt that the AI summaries guided them to first concentrate on high-level informative messages and then narrow them down to single cases.

> *"The most helpful information is the AI summarized issue. That's the first thing I take a look at. The first thing I have is this UI design. So it already helped me filter out the students who are struggling with who have already passed all the test cases. Summarize the issues and I think the*

explanation is also makes some things for me to track. " [P3]

In addition, the AI summarized panel helped participants "speed up" the process of comprehending students' progress and assessing their performance. Visualized clusters, coupled with the AI summarised issues, served as efficient signals to identify common issues and eased the monitoring workload. As P8 said:

> *"So I first go through each category for the performance and click into the history for students, and I see if they have any syntax issues or have used the AI model or the AI help. I can filter out some students. And then I can just focus on the rest. It helps me to speed up my evaluation. So I don't need to revisit the timestamp for each student. I only need to do for some of them." [P8]*

Grounded by these strategies and behaviors, participants not only shifted their monitoring patterns when using ASPIRE, but also improved their productivity while working on large-scale programming submissions.

5.4.3 ***Editable AI Summary Can Empower Instructor Trust.*** Initially, the AI-summarized information was presented to users after it was generation without modification. Despite participants confirming the efficiency of relying on the AI summary to monitor and comprehend students' mental models, the attitude and trustworthiness toward AI-generated information was still under consideration. From the interview, eleven participants expressed positive perspectives about their trust in the generated summary to help them understand students' mental models, while xxx were skeptical about its credibility and accuracy. This doubt increased, especially when participants noticed unclear descriptions and errors, such as mis-classifications, or instances where the generated explanations conflicted with their understanding.

> *"Well, I think I was mostly unable to trust it. Sometimes I think it was good when it came to doing well but, sometimes, it said some of the students that were having some issues were not doing well, even though now I'd say they're all doing fine." [P7]*

Taking into account how participants' mental patterns could be inconsistent with AI summarized information, during the fourth tasks, 11 of the 12 participants explicitly mentioned that they were slightly more trusting of the AI summary after being able to edit it. Participants highlighted how, by integrating personal understanding patterns, the later generated AI summaries became more reliable and gave more comprehensive explanations of students' states.

> *"After I edited the students' performance, I took responsibility to kind of generalize what I said with what it([AI] knew already and it wasn't biased toward my thinking of the student or the AI's thinking. Of course I can trust more because it's not a bias toward one thing." [P10]*

5.4.4 ***Increased Trust For "Doing Well" Students.*** In the ASPIRE condition, we noticed that participants are more likely to trust the AI's perspective when the category is "Doing well" compared to "Getting Stuck" or "Idle." They tended to only check the "Getting Stuck" students' to see if they are struggling as AI suggested.

> *[P7] "I didn't have to worry too much about the students in the "Doing Well" category as every time I did check, the AI was correct in identifying who was making good progress."*

5.5 Limitations and Future Work

One of the main concerns raised by participants was the mistakes made by AI regarding students' categories. As P1 mentioned, "I did not like the fact that the AI could wrongly categorized a student. This is because once me as instructor spot a single students which categorization is wrong I will start checking all the other students in that same group, which could take a while and change my focus from helping other into solving the mistakes from AI." This highlights the need to improving the accuracy of AI's categorization to minimize the time instructors spend on verifying and correcting the AI's assessments. Another limitation identified by the participants was the complexity of the AI-generated explanations. P2 noted "the explanations from AI contain a lot of information and may be hard to grasp in a short time." This suggests that future iteration of the system should explore a better design on communicating AI's decision making criteria. Further more, some participants expressed the concerns about students' privacy. In future deployment, we will allow students to have control over their privacy settings, such as providing the option to disable sharing their chat history with the AI.

6 CONCLUSION

In this work, we introduced ASPIRE, a novel system that enables instructors to ascertain students' coding understanding by attending to their interactions with LLM-based chat bots and tracking student editing activity. It then uses this information to visualize student coding issues and presents a timeline of student interaction to provide instructors with an overview of a student's attempts at solving a coding problem. Our study revealed that ASPIRE aided instructors in identifying activity signals and forming a high-level view of student behaviors. Our work contributes to the augmentation of LLM usage in classroom settings to facilitate an instructor's understanding of student performance beyond pass rate data and code visualization and does so at scale. Overall, this work opens up exciting avenues for the utilization of AI to improve learning at scale.

REFERENCES

[1] 2024. Hume AI. https://www.hume.ai/ Accessed: April, 2024.
[2] Gagan Bansal, Tongshuang Wu, Joyce Zhou, Raymond Fok, Besmira Nushi, Ece Kamar, Marco Tulio Ribeiro, and Daniel Weld. 2021. Does the whole exceed its parts? the effect of ai explanations on complementary team performance. In *Proceedings of the 2021 CHI conference on human factors in computing systems*. 1–16.
[3] Brett A Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming is hard-or at least it used to be: Educational opportunities and challenges of ai code generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 500–506.
[4] Douglas S Blank, David Bourgin, Alexander Brown, Matthias Bussonnier, Jonathan Frederic, Brian Granger, Thomas L Griffiths, Jessica Hamrick, Kyle Kelley, M Pacer, et al. 2019. nbgrader: A tool for creating and grading assignments in the Jupyter Notebook. *The Journal of Open Source Education* 2, 11 (2019).
[5] Yan Chen, Jaylin Herskovitz, Gabriel Matute, April Wang, Sang Won Lee, Walter S Lasecki, and Steve Oney. 2020. Edcode: Towards personalized support at scale for

remote assistance in cs education. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–5.

[6] Nia MM Dowell, Tristan M Nixon, and Arthur C Graesser. 2019. Group communication analysis: A computational linguistics approach for detecting sociocognitive roles in multiparty interactions. *Behavior research methods* 51 (2019), 1007–1041.

[7] Raymond Fok and Daniel S Weld. 2023. In search of verifiability: Explanations rarely enable complementary performance in ai-advised decision making. *arXiv preprint arXiv:2305.07722* (2023).

[8] Katy Ilonka Gero, Chelse Swoopes, Ziwei Gu, Jonathan K Kummerfeld, and Elena L Glassman. 2024. Supporting Sensemaking of Large Language Model Outputs at Scale. *arXiv preprint arXiv:2401.13726* (2024).

[9] Elena L Glassman, Lyla Fischer, Jeremy Scott, and Robert C Miller. 2015. Foobaz: Variable name feedback for student code at scale. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. 609–617.

[10] 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.

[11] 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.

[12] 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.

[13] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. 2023. Exploring the responses of large language models to beginner programmers' help requests. In *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1*. 93–105.

[14] Peiling Jiang, Jude Rayan, Steven P. Dow, and Haijun Xia. 2023. Graphologue: Exploring Large Language Model Responses with Interactive Diagrams. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (<conf-loc>, <city>San Francisco</city>, <state>CA</state>, <country>USA</country>, </conf-loc>) *(UIST '23)*. Association for Computing Machinery, New York, NY, USA, Article 3, 20 pages. https://doi.org/10.1145/3586183.3606737

[15] Shalini Kaleeswaran, Anirudh Santhiar, Aditya Kanade, and Sumit Gulwani. 2016. Semi-supervised verified feedback generation. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 739–750.

[16] Majeed Kazemitabaar, Xinying Hou, Austin Henley, Barbara Jane Ericson, David Weintrop, and Tovi Grossman. 2023. How novices use LLM-based code generators to solve CS1 coding tasks in a self-paced learning environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*. 1–12.

[17] Julia M Markel and Philip J Guo. 2021. Inside the Mind of a CS Undergraduate TA: A Firsthand Account of Undergraduate Peer Tutoring in Computer Labs. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 502–508.

[18] Julia M Markel, Steven G Opferman, James A Landay, and Chris Piech. 2023. GPTeach: Interactive TA training with GPT-based students. In *Proceedings of the tenth acm conference on learning@ scale*. 226–236.

[19] Joseph Bahman Moghadam, Rohan Roy Choudhury, HeZheng Yin, and Armando Fox. 2015. AutoStyle: Toward coding style feedback at scale. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*. 261–266.

[20] Alok Mysore and Philip J. Guo. 2018. Porta: Profiling Software Tutorials Using Operating-System-Wide Activity Tracing. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (<conf-loc>, <city>Berlin</city>, <country>Germany</country>, </conf-loc>) *(UIST '18)*. Association for Computing Machinery, New York, NY, USA, 201–212. https://doi.org/10.1145/3242587.3242633

[21] Andy Nguyen, Christopher Piech, Jonathan Huang, and Leonidas Guibas. 2014. Codewebs: scalable homework search for massive open online programming courses. In *Proceedings of the 23rd international conference on World wide web*. 491–502.

[22] Chris Piech, Jonathan Huang, Andy Nguyen, Mike Phulsuksombati, Mehran Sahami, and Leonidas Guibas. 2015. Learning program embeddings to propagate feedback on student code. In *International conference on machine Learning*. PMLR, 1093–1102.

[23] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*. 27–43.

[24] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. 2003. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 76–85.

[25] Jonathan A Smith. 2015. Qualitative psychology: A practical guide to research methods. (2015).

[26] Sangho Suh, Bryan Min, Srishti Palani, and Haijun Xia. 2023. Sensecape: Enabling Multilevel Exploration and Sensemaking with Large Language Models. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (<conf-loc>, <city>San Francisco</city>, <state>CA</state>, <country>USA</country>, </conf-loc>) *(UIST '23)*. Association for Computing Machinery, New York, NY, USA, Article 1, 18 pages. https://doi.org/10.1145/3586183.3606756

[27] Ahmad Taherkhani, Ari Korhonen, and Lauri Malmi. 2012. Automatic recognition of students' sorting algorithm implementations in a data structures and algorithms course. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*. 83–92.

[28] 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.

[29] Tianjia Wang, Daniel Vargas Díaz, Chris Brown, and Yan Chen. 2023. Exploring the Role of AI Assistants in Computer Science Education: Methods, Implications, and Instructor Perspectives. In *2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 92–102.

[30] Ashley Ge Zhang, Yan Chen, and Steve Oney. 2023. RunEx: Augmenting Regular-Expression Code Search with Runtime Values. In *2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 139–147.

[31] 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.

7 APPENDIX : PROMPTS

Below are the prompts used to generate the AI Summary.

7.1 Generating Summary

System Prompt:

You are a instructor of the python programming class. Current task for student:{Task description with code}

During this process, students can ask questions from GPT.

Definition for "getting stuck": 1)student is making edits but can't get closer to the target solution. 2)Student may be aware of the issue but unable to resolve it, even after seeking AI assistance. 3)Student may not be asking for AI assistance despite being stuck on a problem. 4)Student's pass rate on coding exercises remains consistently low or at 0. 5)Unable to understand error messages or apply the correct fixes. 6)submit code multiple times without making any progress.

Definition for "doing well": 1)student is making edits and getting closer to the target solution. 2)Student is actively engaged in coding and making progress; 3)Student is able to understand and apply the concepts correctly; 4)Student can effectively incorporate the GPT's responses into their code; 5)Student demonstrates a good understanding of programming concepts like indentation, variable initialization, method usage (e.g., append method), etc. 6)Student's pass rate on coding exercises is consistently high or improving. 7)Student is able to understand error messages and apply the correct fixes.

Definition for "idle": 1)Student is not actively engaged in coding or making progress during the current time period. 2)Student has not made any meaningful attempts to solve the problem. 3)Student may be asking unrelated or irrelevant questions, not pertaining to the coding exercise at hand.

Task:

(1) You need to generate a summarization of student performance into three categories: "doing well", "getting stuck", and "idle". Base your summarization on the students' activity data, considering their behavior patterns that I mentioned in the definations rather than just error messages. Provide a short explanation of their performance within 40 words.
(2) And give the issue they are facing right now with 10 words or less.
(3) For each category you assign a student to, provide the reasons and evidence that support your classification: 1)Describe the specific evidence (student actions, code edits, conversations) you based your decision on, including the start and end times if applicable. For a series of continuous edits, you can group them as a single piece of evidence. Summarize each piece of evidence in 10 words or less. 2)For conversations, quote the relevant original text you are referencing. 3)Rate the relevancy of each piece of evidence on a scale of 1 to 10.

Input Format (JSON): Input are the a list of activity data and a list of previous performance of one student format:

```
{"actions":[{"action_type":"edit",
"new_content":"#code after editing#",
"previous_content":"#code before that edit#",
"time_diff":"#time difference from the start#"},
{"action_type":"submission",
"content":"#submitted code#",
"passrate":#passrate for the submission#,
"time_diff":#time difference from the start#,
"output":"#error message student get#"},
{"action_type":"conversation",
"user":"#student's question#",
"GPT":"#GPT's response#",
"time_diff":#time difference from the start#"}...],
"total_submission_attempts":#total number of submissions
                            from start#}
}
```

Output Format (valid JSON): Output format should be json object:

```
{"current_performance":"doing well"/"getting stuck"/"idle"
, "short_explanation":"Brief explanation of performance in
40 words or less", "evidence":
[{"type":"edit","description":"","start_time":,"end_time":,
"relevancy": },
{"type":"conversation","description":"","time":,"content":
["",....],"relevancy": },
{"type":"submission","description":"","time":
,"relevancy":}...], "issue":""}}
```