

(Im)Paired Programming: Coding Agents Improve Productivity but Harm Understanding

Nishant Balepur^{1,2} Connor Baumler¹ Valerie Chen³
Eunsol Choi² Rachel Rudinger¹ Jordan Boyd-Graber¹

¹University of Maryland ²New York University ³Carnegie Mellon University

nbalepur@umd.edu jbg@umiacs.umd.edu

Abstract

Coding agents (e.g., Cursor) improve developer productivity by optimizing task completion, but shifting users from writing code to prompting and reviewing may harm their understanding—impeding oversight, learning, and communication. To probe this, we have 54 students create a website with one of two AI systems: an *agent* that edits user code; or a *chatbot* where users write code alone or adapt generic code snippets. We test understanding via comprehension questions and a task where users extend their code without agents, showing: (1) While agents aid initial task completion, they harm users’ code comprehension and thus do not prepare users to extend their code; (2) Low-effort agent interaction types—like copy+paste prompts and auto-accepted edits—are linked with lower comprehension; and (3) Despite self-reported weaker understanding, users still prefer coding agents because they are quick and easy to use. While users stay in the loop for coding workflows, understanding should not be forgotten. Towards this goal, we distill our analyses into future research directions for coding agent developers: dissuading low-effort prompting, creating readable code, and promoting active engagement.¹

1 Intro: Coding Under the Influence

Coding agents follow natural language queries to modify code for users (Chan et al., 2026, e.g., Cursor). They are increasingly popular in NLP, evinced by the advent of “Code Models” and “LLM Agents” tracks at EMNLP.² Researchers and developers often equate agent progress to task completion metrics in benchmarks (Jimenez et al., 2024), then validate users prefer agents and can use them to finish tasks quickly (Peng et al., 2023; Shao et al., 2026). By this standard, coding agents appear successful.

If coding agent progress aims to mirror real use, task completion is not enough: programmers often

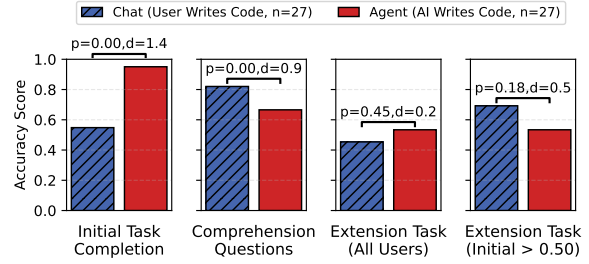


Figure 1: Our user study compares two AIs—an agent that writes code for users and a chatbot that ensures our users write code—for finishing a web development task. Relative to chatbots, agents initially help users complete the task but largely harm comprehension (Cohen, 2013, $p < 0.002$, Cohen’s $d > 0.80$). Agent users can better extend their code without agents, but this reverses when controlling for initial task accuracy, suggesting gains are driven by initial success over improved understanding.

understand their code—knowing its content and behavior (Kam et al., 2025; Qiao et al., 2026). Understanding helps users stay productive if AI is unavailable (Liu et al., 2026, e.g., confidential work, outages), explain their work for others to use (Ko et al., 2007, e.g., research papers, hand-offs), and thwart catastrophic errors (Engels et al., 2026, e.g., database deletion³). Users who write code alone often understand it (Kolb, 1984), but what happens when agents usurp writing and programming turns into prompting and reviewing (Chen et al., 2026)?

This paper tests whether users understand their code after working with coding agents optimized for task completion. 54 CS students make a website in a code editor with one of two prompt-based AI tools: (1) an agent that directly edits the user’s code; and (2) a chatbot that forces users to actively write code from scratch or adapt generic code snippets (Figure 2, left). We then assess user understanding in two ways: comprehension questions tailored to each user’s code (Figure 2, center) and accuracy on a follow-up task where users extend their code to pass new test cases without agents (Figure 2, right).

¹Anonymous repo at: <https://tinyurl.com/anon-coding>

²https://2026.emnlp.org/calls/main_conference_papers/

³Like this unfortunate case: <https://tinyurl.com/claude-DB>

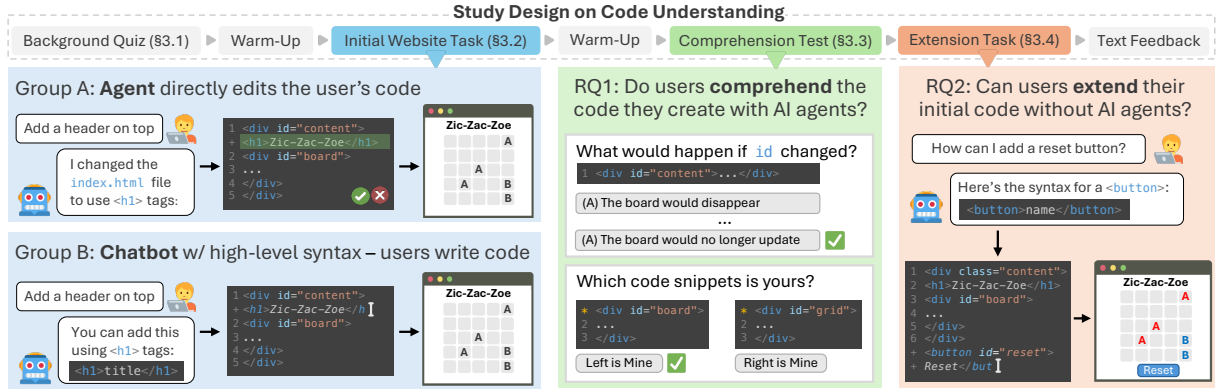


Figure 2: Our between-subjects user study to show how coding agents neglect user understanding. After taking a background quiz, $n = 54$ users design a web-based variant of tic-tac-toe, prompting either: (A) an AI agent that directly edits user’s code; or (B) a chatbot that gives high-level syntax. We test understanding via comprehension questions about the user’s code, and accuracy on a follow-up task where users must extend their code without the agent.

While agent users finish the initial task more accurately, they score substantially lower in comprehension questions and not much higher in the extension task (Figure 1, §4.1). For agent users, novices submit code as accurate as experts but have worse comprehension (§4.2), so traditional coding skills remain valuable. Lastly, we reveal a tradeoff to explain similar extension accuracy (§4.3): agents create higher-quality code at first, but lower comprehension makes the code harder to use. Thus, we argue coding agents today do not aid understanding.

We then qualitatively review user interactions to help explain why agent users struggle to comprehend their code, informing new research directions towards agents that support understanding. Users who rely on lower-effort prompt (copy+paste) and review (auto-accept) strategies have lower comprehension (§5.1, §5.2), motivating techniques for refusing “lazy” requests. Users have higher comprehension when agents write simpler code, so code readability metrics (Buse and Weimer, 2009) could benefit agent training (§5.3). Lastly, we study users’ feedback to find opportunities in routing implementation tasks between users or AI, improving control, and ensuring gains in understanding are clear (§6).

As agents tempt programmers to offload writing, we argue understanding must help shape progress: task completion alone neglects understanding. Toward user understanding of AI code, we contribute:

- 1) An online between-subjects study to assess how coding agents influence the understanding of users.
- 2) Evidence that coding agents improve initial task accuracy, harm understanding, and fail to improve extension accuracy—despite users preferring them.
- 3) Studies of user-agent interaction modes and user feedback to inform the design of AI coding agents.

4) A released dataset of 54 user websites, prompts, AI traces, background ability, and understanding.

2 Background and Related Work

To motivate our study, we discuss how agent evaluations neglect user understanding (§2.1)—a gap software engineering research can help fill (§2.2).

2.1 Evaluating Coding Agents

Coding agents like Codex (Chen et al., 2021), Copilot (Stratton, 2024), Cursor (Chan et al., 2026), and Claude Code (Anthropic, 2026) are increasingly popular. Unlike chatbots (Carreira et al., 2022) and autocomplete (Svyatkovskiy et al., 2020), agents enable workflows where AI writes most code, while the user’s primary role is prompting and reviewing.

Most work develops coding agents to finish tasks (Jimenez et al., 2024; Merrill et al., 2026), leading researchers to ask if this metric fully supports users in practice (Mozannar et al., 2025). Prior evaluations show offline metrics convert to online benefits for users’ productivity (Peng et al., 2023; Cui et al., 2026),⁴ but add new risks in security (Wahed et al., 2025; Wang et al., 2025) and long-term maintainability (Ottenhof et al., 2026; Huang et al., 2026).

Evaluating code understanding with agents is unexplored: a risk that impedes users’ knowledge and use of their code (Wyrich et al., 2023). The most similar work shows agents optimized for task completion harm learning of new skills (Kazemitabaar et al., 2023; Shen and Tamkin, 2026), but understanding is distinct: learning tests if users acquire generalizable skills, but understanding tests if users can reason about and work with their *own* code.

⁴Becker et al. (2025) is an exception, but they recognize flaws such as selection bias in evaluation (Becker et al., 2026).

2.2 Measuring Code Understanding

To develop metrics for user understanding with coding agents in our study, we draw on software comprehension research (Brooks, 1983)—a field dedicated to how users reason about code. Prior work tests understanding by asking users to recall (Shepard et al., 1979), trace (Peitek et al., 2018), debug (Arunachalam and Sasso, 1996), and extend code snippets (Besker et al., 2020), or self-reporting it (Boehm et al., 1976). We synthesize these metrics for our user study, using comprehension questions about users’ code (§3.3), accuracy on an extension task (§3.4), and self-reported user feedback (§6).

Most work studies code understanding outcomes after manipulating aspects of pre-defined snippets (Wylich et al., 2023), such as readability (Johnson et al., 2019), complexity (Peitek et al., 2021), and structure (Wiese et al., 2019). Studying if users understand their own code is rare (Dasgupta, 2010)—as writing code is often assumed to aid understanding (Kolb, 1984)—but is now at risk with agents replacing writing. Most similar in this field, Seo et al. (2026) test if users feel “ownership” of their code with AI. We instead design a study to reveal agent users struggle to understand their code (Figure 1).

3 Study Design: AI Website Development

Bridging research on agent evaluation and program comprehension (§2), we run a user study (Figure 2) to test productivity and understanding with coding agents. We focus on web development, one of the most common coding agent use cases (Anthropic, 2025). Users take a background quiz (§3.1), then we test productivity when users have agents write code versus mostly write code by themselves (§3.2). We finally instantiate code understanding metrics: users answer comprehension questions about their code (§3.3) and extend their initial code (§3.4). Below, we outline each step, then recruitment (§3.5).

3.1 Control for Analyses: Background Quiz

We first measure background (BG) ability as a control for understanding (Siegmund and Schumann, 2015). Since we study website tasks (§3), we use multiple-choice HTML, CSS, and JavaScript questions from LinkedIn skill quizzes,⁵ popular for interview preparation.⁶ We curate questions to target

⁵<https://github.com/Ebazhanov/linkedin-skill-assessments-quizzes>

⁶Ideally, we would use a validated concept inventory (Cac-effo et al., 2016), but they do not exist for web development.

four skills in each programming language—syntax recall, conceptual knowledge, code tracing, and implementation (Xie et al., 2019)—for 12 total items (Appendix A.8.1). A front-end engineer validated questions. Question accuracy measures BG ability.

3.2 Productivity Metric: Initial Website Task

We use one task across all users for controlled analyses, a standard user study design (Yan et al., 2024; Shen and Tamkin, 2026). We construct an original game creation task—a variation of tic-tac-toe called “zic-zac-zoe”—where users modify three files for page structure (HTML), game logic (JavaScript), and style (CSS). Thus, it is complex enough to test understanding while staying familiar. We now outline the task, AI groups, and productivity metrics.

Task Description: In zic-zac-zoe, users start with a blank 5x5 board and must implement the following:

1. Turns alternate between Player A/B, placing two symbols per turn ($A \rightarrow A \rightarrow B \rightarrow B \rightarrow \dots$).⁷
2. When a player clicks on an unoccupied square, it should display their symbol (“A” or “B”).
3. The game is over when a player occupies an entire row or column (not the diagonals), or no more moves can be played (i.e., a tie game).
4. The CSS should center the page horizontally.
5. A status element must show the next turn (i.e., “Player A Turn” or “Player B Turn”) and a message when the game ends (e.g., “A Wins!”).

All users view criteria in text/video format and have a 50 min. time limit, based on the time taken in a pilot study with three Ph.D. students. Users can read all criteria before starting and submit early if done.

We only tell users to implement the above criteria; we do not mention that they will later answer comprehension questions or extend their code.

AI Groups: We adapt VibeJam’s agentic coding UI (Balepur et al., 2026, Figure 3), where users view the task description and their website (left), write code in a VSCode-like IDE with HTML/CSS/JS starter files (middle), and prompt AI for help (right).

To compare understanding when users actively write code or offload writing, we randomly assign users one of two prompt-based AI using GPT-4.1:

1. Group A has an **agent** that directly edits code (Figure 3, a). We use the open-source AIDER agent; it executes prompts by proposing edits as diffs on users’ code they can accept/reject, and summarizes executions (Gauthier, 2023).

⁷Like tic-tac-toe (Beck, 1945), we believe zic-zac-zoe is unwinnable under optimal play, but still valid for our purposes.

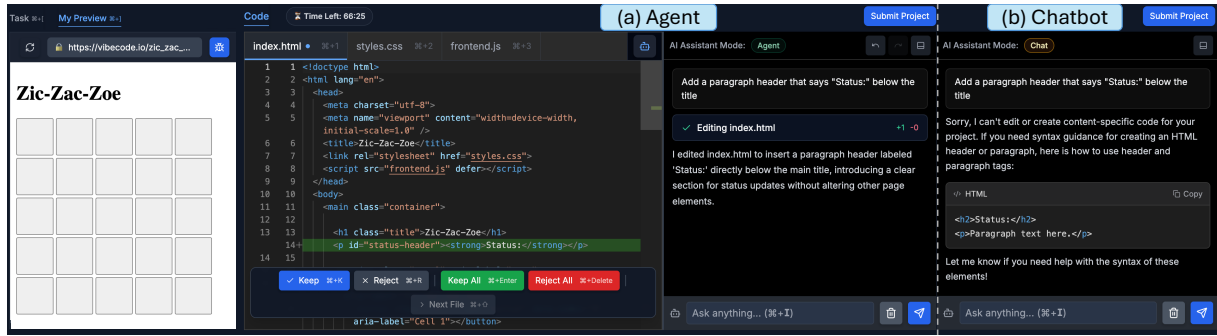


Figure 3: Overview of UI and AI groups. Users create a website that meets input requirements (left) by prompting one of two AI systems: (a) an agent that edits users code, explains edits, and affords user review; or (b) a chatbot that gives high-level syntax.

2. Group B has a **chatbot** that only gives generic code snippets, so users must write code from scratch or manually adapt snippets (Figure 3, b). This mirrors how programmers adapt code (e.g., StackOverflow) while keeping the same prompt interactions as the agent. It also limits non-compliance rife in no-AI studies (Zhang et al., 2025, e.g., using ChatGPT). The chatbot is an LLM without access to users’ code. Its prompt rejects queries beyond syntax help and instructs to only generate short text or generic code blocks of five or less lines (Prompt A.1).

Productivity Metric: We derive atomic rules from task criteria (e.g., “*The CSS should center the page horizontally*” → “*The page is centered horizontally*” and “*The page is centered via CSS*”), forming a 15-rule rubric (Rubric A.1). We hide this rubric from users to limit gaming (Baker et al., 2008). We score productivity as code accuracy—the proportion of met rubric rules—and time-to-submission for secondary analyses. Following Sharma et al. (2026), Gemini-3.1 Pro judges user code with our rubrics (Prompt A.6). To validate, Author A and B score ten agent and ten chatbot users’ code via the rubric; they agree in 98% of rules. Gemini has near-perfect Cohen’s κ of 0.96 with Author A (Cohen, 1960).

3.3 Understanding Metric 1: Comprehension

Software engineering research often uses comprehension questions to assess understanding (Wyrich et al., 2023), but this is hard for code users write: you must ask about a user’s code in real time. Thus, our first understanding metric has LLMs tailor comprehension questions to users’ code linked to two Bloom’s taxonomy levels (Krathwohl, 2002): 1) if users can recall elements in their code; and 2) how users reason about their code behavior. Users also self-report productivity, understanding, and preferences during zic-zac-zoe, which we study in §6.1.

We use six questions of each type—two HTML, CSS, and JS—for 12 in total. After the initial task, users answer these *without* seeing their code, apart from snippets we show. Accuracy on comprehension questions is our first understanding metric (Figure 1, middle). We explain the question types next.

Recall: We test how well users identify their code (Sheppard et al., 1979), predictive of memory/learning (Ebbinghaus, 1913; Zindulka et al., 2026). We ask: (1) three multi-select questions where users select which HTML elements, CSS selectors, and JS functions exist in their code (Appendix Figure 15); and (2) three comparisons where users pick which of two logically-equivalent JS/HTML/CSS code snippets exist (Appendix Figure 16). We randomly sample parts of users’ code that they had to modify in the initial task for ground-truth answers and use GPT-5.2 to create distractors (Prompts A.3, A.4).

We validate 30 questions from pilot submissions for accuracy (e.g., answers exist in code). To check for stylistic cues in LLM distractors, two students answer 12 questions without seeing the code they are based on. They achieve 0.38 accuracy (random is 0.5), so questions are not easily guessable from the choices alone (Lau et al., 2011). Appendix A.9 validates this holds for real questions in our study.

Reasoning: We design multiple-choice questions to probe how users reason about their code (Peitek et al., 2018). Each question asks users about one of three main code snippets they had to write, each linked to a task criterion in §3.2: HTML for status, CSS selector for page centering, and JS function for board rendering.⁸ For each snippet, we ask two questions prior work suggests users can solve only if they understand the code: (1) its *purpose* (Denny et al., 2024, Appendix Figure 17); and (2) how the website would *change* if the snippet changed in a

⁸If the user did not attempt a specific criteria, we do not ask questions about it; GPT-5.2 checks for valid user attempts.

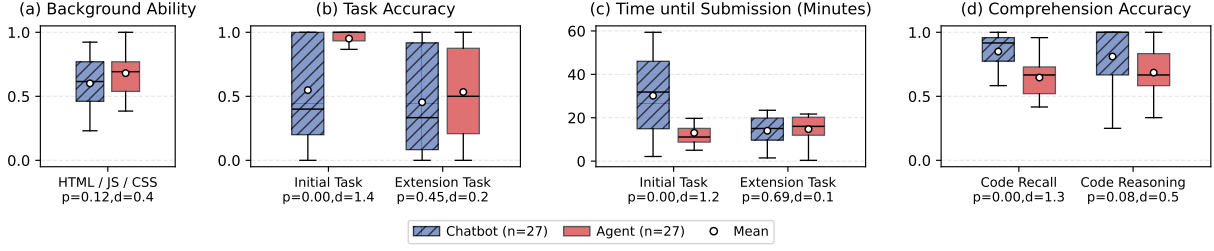


Figure 4: Metrics over AI groups. Despite similar background ability (a), agent users are more accurate/quick in the initial task (b/c, left), but not when extending code without agents (b/c, right) and agent users have worse comprehension (d). Appendix A.9.2 shows poor comprehension persists for 3/4 question types (all but *change* questions). Coding agents do not support understanding.

given way (Zeller, 2009, Appendix Figure 18). To control for content and tailor questions to users, we design 12 question templates, then prompt GPT-5.2 to infill them using the user’s code (Prompt A.5).

A front-end engineer reviewed templates and Author A ran five faulty and five perfect submissions to validate GPT aptly infills templates. This holds for real questions during our study (Appendix A.9).

3.4 Understanding Metric 2: Extension Task

We lastly test how well users edit their code without extensive AI support, a downstream metric of code understanding (Lucas et al., 2019; Nielebock et al., 2019). Users start with their code submitted in §3.2 and must extend it to pass the new criteria below:

1. Add another win condition where players can win by occupying all four corners of the board.
2. Style the board symbols in the CSS file so that “A” symbols are red and “B” symbols are blue.
3. Add a “Reset” button to reset the entire game.

All users finish this task with the chatbot. Like §3.2, no-AI is difficult to enforce, but the chatbot still requires users to actively write code—testing extension skill when users cannot offload writing to AI.

We use the same UI as §3.2 and give 20 minutes, based on the time taken in pilot testing. Agent users first finish a five minute warm-up task to familiarize with the chatbot. We score task accuracy on a 12-rule rubric (Rubric A.2) via the same LLM judge in §3.2. After submission, we ask open-ended questions about their experiences, which we study in §6.

3.5 User Recruitment

We recruit 59 CS students in two U.S. colleges (17 B.S., 31 M.S., 11 Ph.D.), as such users code with AI (Vadaparty et al., 2024) and need to understand their code in coursework or research (Kazemitabaar et al., 2025). Users report 5.4 ± 0.3 years of coding experience in any language and 16.2 ± 1.4 months coding with AI (e.g., ChatGPT, Cursor). We add attention checks in background (§3.1) and compre-

hension quizzes (§3.3), and drop data of five users who fail them. Users prompt 5.8 ± 0.4 (agent) and 11.6 ± 2.5 (chatbot) times in the initial task, and 6.8 ± 0.9 (agent) and 4.7 ± 0.6 (chatbot) times in the extension task. Course instructors give extra credit for taking the study, approved by IRB (Ethics §9).

4 Results: Outcomes with Coding Agents

We now study our metrics to expose agent users are productive but lack understanding (§4.1). We then show agents mask background skill (§4.2) and give useful code for users initially, but users’ worse comprehension makes this code harder to extend (§4.3).

4.1 Coding Agents Harm Understanding

We first compare our productivity and understanding metrics across AI groups. Agents users finish the initial task more accurately/quickly than chatbot users, but these gains do not largely transfer to the extension task (Figures 1, 4). Agent users score much lower on recall and reasoning questions, despite access to mechanisms meant for understanding: diff reviews and summaries of execution traces. Overall, prompting and reviewing agent code did not support understanding as well as writing code (Shen and Tamkin, 2026). We return in §5 to study how to design AI that improves user understanding.

4.2 Background Skill Drives Comprehension

We now test how user background (BG) influences understanding in our study. We use linear regression (Fisher, 1922) to predict user outcomes y_{out} (initial/extension accuracy, comprehension) via BG ability x_{bg} , AI group x_{group} , and their interaction:

$$y_{out} \sim \beta_0 + \beta_1 \cdot x_{group} + \beta_2 \cdot x_{bg} + \beta_3 \cdot x_{bg} \times x_{group}, \quad (1)$$

where $x_{group} = 1$ for chatbot and 0 for agent users. By setting $x_{group} = 0$, we can use β_2 to capture how BG ability influences user outcomes with agents, and for $x_{group} = 1$, we can use $\beta_2 + \beta_3$ to capture how BG ability influences outcomes with chatbots.

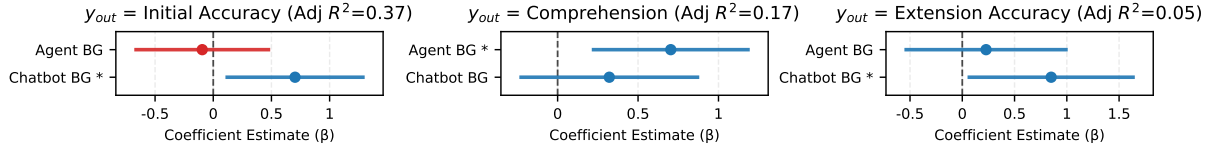


Figure 5: Regression coefficients with 95% CIs when predicting study outcomes from user background (BG) ability, segmented by AI groups. We show: (1) BG ability only predicts initial task accuracy in chatbot users, with agents equalizing users regardless of BG; (2) BG strongly predicts comprehension across groups; and (3) BG weakly predicts extension accuracy (Adj. $R^2 = 0.05$).

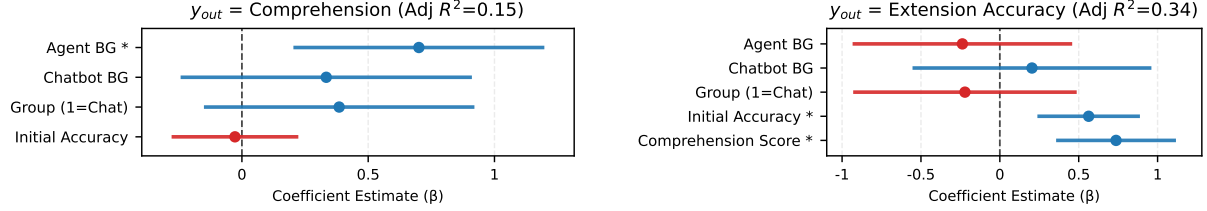


Figure 6: Regression coefficients with 95% CIs after adding initial task accuracy to the comprehension regression (Equation 1). Initial accuracy barely improves adjusted R^2 ($0.40 \rightarrow 0.42$), so comprehension is distinct from task completion.

Figure 7: Regression coefficients with 95% CIs after adding initial task accuracy to the extension accuracy regression (Equation 1). Both feature coefficients are significantly positive and largely boost R^2 ($0.05 \rightarrow 0.22$), showing why agent/chatbot users have similar extension accuracy: agents trade better code scaffolds for worse comprehension, impeding code extension.

Higher BG ability users have higher initial task scores with chatbots but not agents (Figure 5, left)—all users score highly—so agent use can mask how evaluation settings (i.e., classrooms, hiring) discern users with stronger BG (Xu et al., 2025). Comprehension scores positively predict BG ability (Figure 5, middle), so traditional coding BG is valuable for understanding even with agents (Thorgeirsson et al., 2026). Lastly, higher BG predicts extension accuracy in chatbots but not agents (Figure 5, right), so experts and novices may struggle similarly when extending their code without agents after agent use.

4.3 Agents Trade Accuracy for Understanding and this can Degrade Extension Ability

To learn what drives understanding, we add features to our regression (Eq. 1): 1) initial task accuracy to predict comprehension; and 2) initial task accuracy and comprehension to predict extension accuracy.

Initial task accuracy has a small, insignificant coefficient for comprehension predictions (Figure 6), showing that task completion and comprehension are distinct optimization goals. The regression also reveals that agent users still have significantly lower comprehension; thus, it is unlikely that this difference stems from LLMs producing easier questions for the chatbot users who did not complete the task.

Initial task accuracy and comprehension scores positively relate to extension accuracy (Figure 7), with large adjusted R^2 gains from the initial model ($R^2 = 0.05 \rightarrow 0.34$). This helps explain similar extension accuracy over AI groups (Figure 1): agents generate higher-quality code scaffolds but sacrifice comprehension, impairing users’ website extension

when agents disappear (Collins et al., 1988). A path mediation model confirms this in Appendix A.5.

5 Why Agent Users Lack Comprehension

Agents harm comprehension (§4.1)—which in turn may limit extension ability without agents (§4.3)—so we now propose ways to design agents that aid comprehension. We study steps of our agent workflow: prompting (§5.1) and reviewing (§5.2) agents, and attributes of the code (§5.3)—concluding each with design takeaways for future work. Our sample size ($n = 27$) matches Shen and Tamkin (2026) but is still moderate, so all analyses are exploratory.

5.1 Discourage “Lazy” Prompting

Prompting affects learning (Srinath et al., 2025), so we test how this shapes comprehension via qualitative coding (Bingham, 2023). Author A reviews ten users’ prompts and after two rounds, discovers four high-level strategies: directly copying task criteria, turning criteria into code syntax, iterative debugging, and eliciting explanations (Table 1). We then use Gemini-3 Flash to label a strategy for all 152 prompts (Prompt A.7). To validate, Author A and B each label 50 random prompts, agreeing in 90% of cases; Gemini and Author A agree in 94% of cases.

Takeaway 1: Users who copied criteria had the lowest comprehension and users adding code syntax had the highest, despite similar BG (Table 1). This motivates the possibility for refusal (Bai et al., 2022): if prompts use low-effort strategies (copying), agents could nudge users to review their code

Prompting Strategy	Comp. (↑)	BG Ability	# Users	IP(Used)
Asking the agent to explain the codebase (“First, explain the whole project”)	0.635	0.692	6	0.046
Copying criteria (“Add a status element in a paragraph tag...”)	0.654	0.668	25	0.793
Iterative debugging (“Now there is a problem where it only shows player B...”)	0.692	0.662	5	0.040
Turning criteria into syntax (“Broadcast <u>statusMessage</u> to a <u>p</u> tag under the <u>h1</u> tag”)	0.743	0.684	9	0.087

Table 1: Agent prompt strategies and the mean comprehension of the users who use them. Users who added code syntax to prompts had the best comprehension. Regressions support this trend (Appendix A.4). Appendix A.10 analyzes chatbot prompts.

Interaction	Comp. (↑)	BG Ability	# Users	IP(Used)
Auto-Accept	0.615	0.603	6	0.053
Override Changes	0.661	0.682	15	0.318
Hit Accept (All)	0.664	0.669	20	0.492
Hit Accept (Each)	0.777	0.714	7	0.136

Table 2: Mean comprehension of users who engage in different AI code review. Users who auto-accept agent changes have lower comprehension versus users who review each file.

and rewrite their prompt in technical terms before proceeding. This requires advances in classifiers to flag low-effort prompts (Song et al., 2007) and post-training methods to imbue refusal in agents.

5.2 Actively Engage Users During Review

Agent users mostly interact with their code by reviewing AI edits (Kashif et al., 2025), but effort can vary; when agents edit code, users can: 1) prompt again to auto-accept changes; 2) accept all changes by clicking “Accept All”; 3) accept changes individually by clicking “Accept” per file; or 4) override the changes via manual edits or clicking “Reject”.

Takeaway 2: Agent users who review each file have higher comprehension than those who auto-accept edits (Table 2)—and tend to have higher BG scores—but even these users do not reach chatbot users’ mean comprehension (Figure 1). For comprehension, current reviewing interactions may never replace writing code (Kolb, 1984). Thus, NLP+HCI research has a gap in designing agent interactions that encourage active engagement versus passive review, boosting comprehension (Yan et al., 2025).

5.3 Generate More Readable Code

We study how the type of code agents create relates to comprehension. We focus on code readability—known to influence comprehension (Oliveira et al., 2020)—and analyze agent users’ final JavaScript file, as it hosts most of zic-zac-zoe’s logic (§3.2).

We adopt standard readability metrics from Posnett et al. (2011): **1) lines of code**; **2) entropy**—how evenly tokens⁹ (e.g., variables) are distributed (high entropy has more variety); and **3) volume**—

⁹We use the Esprima tokenizer: <https://esprima.org/>. Appendix A.3 has implementation details and metric formulas.

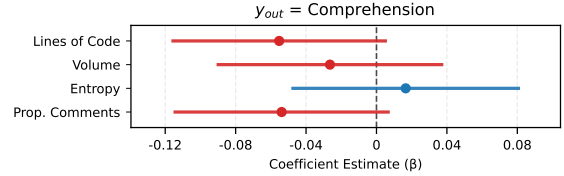


Figure 8: Model coefficients and 95% CIs when using z-score normalized code readability metrics to predict agent user comprehension, fitting four regressions. Users with fewer lines of code and proportion of comments have better comprehension.

total information via unique token count and (1). As documentation shapes comprehension (Brooks, 1982), we also measure the **Proportion** of lines with **Comments**. We fit four regressions—predicting comprehension from each readability metric—as metrics have high colinearity (Posnett et al., 2011).

Takeaway 3: Users with JS code of fewer lines, comments, and volume had higher comprehension (Figure 8), suggesting agent users better digest concise code without distracting comments—observed in past AI coding interviews (Barke et al., 2022). As code properties may shape understanding, this motivates optimizing agents to follow best readability practices (Maalej et al., 2014) beyond just task completion (Le et al., 2022). This spurs challenges in using these metrics for training (Jain et al., 2025) while curbing reward hacking (Zhong et al., 2026).

6 How Users Perceive Coding Agents

To complement our analyses of observed user behavior (§5), we now examine users’ self-reported (§6.1) and written feedback (§6.2) to derive more directions for future NLP research in coding agents.

6.1 Users Recognize Reduced Understanding with Coding Agents, but Still Prefer Them

We review users’ self-reported productivity, understanding, and preferences from §3.3. Users find the agent more helpful and simpler than chatbots, but recognize weaker understanding (Table 3). Despite this, users prefer the agent over chatbots—but most prefer having the option to switch between the AIs.

Takeaway 4: Users may prefer AI that optimizes for task completion, so improving understanding

From 1–5, how much do you agree:	Agent	Chatbot
The agent/chatbot was helpful	4.7 ± 0.2	3.3 ± 0.2
The agent/chatbot took mental effort	1.9 ± 0.2	3.7 ± 0.2
I reviewed agent/chatbot responses	3.2 ± 0.3	4.0 ± 0.2
The code I submitted feels like my own	2.5 ± 0.2	3.7 ± 0.2
I understand how my code works	3.3 ± 0.3	4.2 ± 0.2
I could easily extend my code	3.1 ± 0.3	3.2 ± 0.2
I prefer no agent/chatbot in new tasks	1.4 ± 0.2	1.9 ± 0.2
I prefer the chatbot in new tasks	3.2 ± 0.3	2.7 ± 0.2
I prefer the agent in new tasks	4.2 ± 0.2	3.9 ± 0.2
I prefer switching between agent/chatbot	4.6 ± 0.2	4.1 ± 0.3

Table 3: User 1–5 ratings on usefulness, understanding, and preferences across agent and chatbot groups. The group with the highest usefulness and understanding are **bold**, and the most favored AI is **bold**. Users report the agent as more helpful and easier to use than the chatbot but recognize it harms understanding. Still, users prefer using the agent. Appendix A.6 shows these perceptions correlate with true understanding.

cannot rely solely on promoting alternative systems like chatbots. However, as users prefer switching between AI, future studies can assess whether they do so responsibly, or design routers to predict when AI or users should implement code to balance productivity and understanding (Miranda et al., 2025).

6.2 Why Users Feel They Lack Understanding

Lastly, to learn why our users believe coding agents impair understanding, we review their written feedback from §3.4, distilled into four common themes.

Who Needs to Think. Coding with agents required little mental effort. Many users conveyed “*I didn’t have to think at all.*” (U10), especially since they did not need to use website or CS knowledge (U15, U49) and the agent had context from the users’ files (U31), allowing users to prompt “*fuzzier instructions*” (U35). As users could preview their website, they felt no desire to review changes (U27, U54). U46 rarely reviewed, despite knowing it would be “*detrimental in the long run*”, showing the need for lightweight interactions to promote active thinking.

Agents Can Get Out of Control. Guiding agent behavior was hard. U16 said agents would “*change somewhere unexpected or even destroy the original functions*”, impairing understanding. In U3’s submission, the agent eagerly included features not needed in the task (e.g., board styling), further obscuring the code. Future work can explore methods to add user control to execution (Feng et al., 2026) and ensure the execution trace summaries faithfully explain all changes agents make (Li et al., 2025).

Make Code Quality Actionable. Users found extension tasks easy when agents wrote high-quality

code, citing comments (U44, U54), modular structure (U5, U46), and well-named functions (U21)—showing the benefits of rewarding these signals beyond task completion (Jimenez et al., 2024). However, these benefits resembled onboarding versus better understanding: “*Comments made it easy to understand where to add new features, but having not fully reviewed the code made it much harder to understand everything*” (U31). Thus, code quality alone is insufficient; agents will need to guide users through output code to truly support understanding.

Understanding is Polarizing. While many professional developers value understanding in AI coding (Kam et al., 2025), users were split. U57 wished the agent “*made me understand where code is supposed to go*”, but most praised agents’ accuracy and speed (U5, U26, U54). For chatbot users, U28 felt it “*really helped remember how to use [the code] later on*”, but others rejected it: U5 said “*It’s totally useless*” and U49 noted its value for learning, but asked: “*What’s the point of learning all this when the AI can build it out for you?*” A core challenge in agent design is making benefits of understanding explicit, rather than assuming users will value it.

7 Conclusion: What’s Next for AI Agents

Task completion defines progress for coding agents (Shen et al., 2025), but this goal alone erodes user understanding. While we expose this empirically in our zic-zac-zoe website task, work remains in generalizing our understanding study across programming tasks like algorithmic design (Cormen et al., 2022) and data analysis (Chambers, 1998), testing longitudinal effects via comprehension quizzes every n projects (Kosmyna et al., 2025), and whether existing interactions for coding agents (Feng et al., 2026, e.g., plan mode, switching AI modes) mitigate these issues. Our open-sourced study design, interface, and data will support these future efforts.

Beyond diagnosis, our future work aims to teach agents to support understanding. Our analyses motivate jointly optimizing AI for task completion and understanding signals based on code trace data (Chi et al., 2026), improving faithfulness in execution summaries (Zhang et al., 2023), generating readable code (Jin et al., 2026), and more engaging AI review mechanisms (Sharma et al., 2024). Agents might soon automate many coding tasks, but if we want users to work with their code—monitoring for failures, writing research papers, and learning new skills—agents cannot neglect user understanding.

8 Limitations

Our study is limited in population and domain: we evaluate CS students on two website development tasks. While common in NLP+HCI user studies with coding agents (Yan et al., 2024; Shen and Tamkin, 2026), these findings may not apply to other user groups and tasks. Thus, we encourage future work to extend our protocol to other domains; we open-source all our data and UI to facilitate these efforts.

Further, developers typically have custom workflows in practice (Chen et al., 2026) but for feasibility and to control agent use, our interface supports fewer workflow customization features. Future research could test whether our results hold for users working with coding agents they choose—such as Cursor (Chan et al., 2026) or Codex (Chen et al., 2021)—by using our LLM-tailored comprehension questions with proctored sessions (Thorgeirsson et al., 2026) or IDE plugins (Yan et al., 2024).

To ease the burden of manually scoring code and creating questions per user, we rely on imperfect NLP tools. Despite our checks—discussions with a front-end engineer, manual review of all generated questions, and LLM validation with author-labeled websites—errors still may have slipped through; in Appendix A.9, we more rigorously analyze LLM-generated questions and do not detect major issues. Future research can explore standardizing and rigorously evaluating LLMs to support user study design and analysis, such as qualitative coding tools (Lam et al., 2024), question generators (Anugraha et al., 2026), and rubric-based judges (Kim et al., 2024).

Finally, while we argue that user understanding is traditionally a desired objective for programming agents in real-world workflows, we acknowledge there are cases that short-term productivity may be sufficient, such as personal projects just for fun (Hu et al., 2026, e.g., vibe coding a personal website). However, many novices attempt these projects to gain realistic programming experience—promoting learning, ownership, and engagement (Zhang and Ma, 2023). As a result, coding agents that jointly improve productivity and understanding would still help these users—and should at least be an option.

9 Ethical Considerations

Our study shows that coding agents allows users to finish tasks without comprehending the code they write. If this persists over longer periods, this could impede and possibly weaken user’s skill formation (Shen and Tamkin, 2026). Future work should con-

sider these risks when designing coding agents, and we outline many new directions so NLP researchers can design tools that encourage user understanding.

For our specific user study, we do not collect and release PII in our data and compensate our students with coursework credit. To ensure students are not coerced into participating in our study, we provide an additional assignment that users can complete for the same extra credit—implementing a research paper in Python—that requires the same amount of time (~ 2 hours) as our study. This compensation structure was approved by our organization’s IRB.

We used Generative AI (GenAI) in this project. We used Cursor¹⁰ to design our interface and plots, and ChatGPT to refine paper writing. GenAI did not directly write any parts of this paper. We take responsibility for GenAI errors. By discussing AI use here, we encourage NLP researchers to do the same.

References

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Anthropic. 2025. [Anthropic economic index: Ai’s impact on software development](#).
- Anthropic. 2026. [2026 agentic coding trends report: How coding agents are reshaping software development](#).
- David Anugraha, Vishakh Padmakumar, and Diyi Yang. 2026. [Sparkme: Adaptive semi-structured interviewing for qualitative insight discovery](#). *Preprint*, arXiv:2602.21136.
- Vairam Arunachalam and William Sasso. 1996. Cognitive processes in program comprehension: An empirical analysis in the context of software reengineering. *Journal of Systems and Software*, 34(3):177–189.
- Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, and 12 others. 2022. [Training a helpful and harmless assistant with reinforcement learning from human feedback](#). *Preprint*, arXiv:2204.05862.
- Ryan Baker, Jason Walonoski, Neil Heffernan, Ido Roll, Albert Corbett, and Kenneth Koedinger. 2008. Why students engage in “gaming the system” behavior in interactive learning environments. *Journal of Interactive Learning Research*, 19(2):185–224.

¹⁰<https://cursor.com/agents>

- Nishant Balepur, Connor Baumler, Valerie Chen, Eunsol Choi, Rachel Rudinger, and Jordan Lee Boyd-Graber. 2026. [Vibejam: An open platform for user studies on agentic vibe coding](#).
- Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2022. [Grounded copilot: How programmers interact with code-generating models](#). *Proceedings of the ACM on Programming Languages*, 7:85 – 111.
- József Beck. 1945. Combinatorial games. *American history*, 1861(1900).
- Joel Becker, Nate Rush, Elizabeth Barnes, and David Rein. 2025. [Measuring the impact of early-2025 ai on experienced open-source developer productivity](#). *Preprint*, arXiv:2507.09089.
- Joel Becker, Nate Rush, Tom Cunningham, David Rein, and Khalid Mahamud. 2026. We are changing our developer productivity experiment design. <https://metr.org/blog/2026-02-24-uplift-update/>.
- Terese Besker, Hadi Ghanbari, Antonio Martini, and Jan Bosch. 2020. The influence of technical debt on software developer morale. *Journal of Systems and Software*, 167:110586.
- Andrea J Bingham. 2023. From data management to actionable findings: A five-phase process of qualitative data analysis. *International journal of qualitative methods*, 22:16094069231183620.
- Barry W Boehm, JR Brown, and M Lipow. 1976. Quantitative evaluation of software quality. *Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research*, page 25.
- Ruven Brooks. 1982. A theoretical analysis of the role of documentation in the comprehension of computer programs. In *Proceedings of the 1982 conference on Human factors in computing systems*, pages 125–129.
- Ruven Brooks. 1983. Towards a theory of the comprehension of computer programs. *International journal of man-machine studies*, 18(6):543–554.
- Raymond PL Buse and Westley R Weimer. 2009. Learning a metric for code readability. *IEEE Transactions on software engineering*, 36(4):546–558.
- Ricardo Caceffo, Steve Wolfman, Kellogg S. Booth, and Rodolfo Azevedo. 2016. [Developing a computer science concept inventory for introductory programming](#). In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education, SIGCSE '16*, page 364–369, New York, NY, USA. Association for Computing Machinery.
- Gustavo Carreira, Leonardo Silva, Antonio Jose Mendes, and Hugo Goncalo Oliveira. 2022. Pyo, a chatbot assistant for introductory programming students. In *2022 international symposium on computers in education (SIIE)*, pages 1–6. IEEE.
- John M Chambers. 1998. *Programming with data: A guide to the S language*.
- Aaron Chan, Ahmed Shalaby, Alexander Wettig, Aman Sanger, Andrew Zhai, Anurag Ajay, Ashvin Nair, Charlie Snell, Chen Lu, Chen Shen, Emily Jia, Federico Cassano, Hanpeng Liu, Haoyu Chen, Henry Wildermuth, Jacob Jackson, Janet Li, Jediah Katz, Jiajun Yao, and 35 others. 2026. [Composer 2 technical report](#). *Preprint*, arXiv:2603.24477.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. [Evaluating large language models trained on code](#). *Preprint*, arXiv:2107.03374.
- Valerie Chen, Ameet Talwalkar, Robert Brennan, and Graham Neubig. 2026. [Code with me or for me? how increasing ai automation transforms developer workflows](#). In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems, CHI '26*, New York, NY, USA. Association for Computing Machinery.
- Wayne Chi, Valerie Chen, Ryan Shar, Aditya Mittal, Jenny Liang, Wei-Lin Chiang, Anastasios Nikolas Angelopoulos, Ion Stoica, Graham Neubig, Ameet Talwalkar, and Chris Donahue. 2026. [Editbench: Evaluating LLM abilities to perform real-world instructed code edits](#). In *The Fourteenth International Conference on Learning Representations*.
- John Joon Young Chung, Vishakh Padmakumar, Melissa Roemmele, Yi Wang, Yuqian Sun, Tiffany Wang, Shm Garanganao Almeda, Brett A. Halperin, Yuwen Lu, and Max Kreminski. 2025. [Literarytaste: A preference dataset for creative writing personalization](#). *Preprint*, arXiv:2511.09310.
- Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46.
- Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences*. routledge.
- Allan Collins, John Seely Brown, and Susan E Newman. 1988. Cognitive apprenticeship: Teaching the craft of reading, writing and mathematics. *Thinking: The Journal of Philosophy for Children*, 8(1):2–10.
- Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.
- Lee J Cronbach and Paul E Meehl. 1955. Construct validity in psychological tests. *Psychological bulletin*, 52(4):281.
- Kevin Zheyuan Cui, Mert Demirer, Sonia Jaffe, Leon Musolff, Sida Peng, and Tobias Salz. 2026. The effects of generative ai on high-skilled work: Evidence

- from three field experiments with software developers. *Management Science*.
- Chandan Dasgupta. 2010. That is not my program: Investigating the relation between program comprehension and program authorship. In *Proceedings of the 48th annual ACM Southeast Conference*, pages 1–4.
- Paul Denny, David H Smith IV, Max Fowler, James Prather, Brett A Becker, and Juho Leinonen. 2024. Explaining code with a purpose: An integrated approach for developing code comprehension and prompting skills. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. I*, pages 283–289.
- Dimiter M Dimitrov and Phillip D Rumrill, Jr. 2003. Pretest-posttest designs and measurement of change. *Work*, 20(2):159–165.
- Hermann Ebbinghaus. 1913. *Memory: A contribution to experimental psychology*. Teachers College Press, New York.
- Joshua Engels, David D. Baek, Subhash Kantamneni, and Max Tegmark. 2026. *Scaling laws for scalable oversight*. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- K. J. Kevin Feng, Kevin Pu, Matt Latzke, Tal August, Pao Siangliulue, Jonathan Bragg, Daniel S Weld, Amy X. Zhang, and Joseph Chee Chang. 2026. *Co-coa: Co-planning and co-execution with ai agents*. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI ’26, New York, NY, USA. Association for Computing Machinery.
- Ronald A Fisher. 1922. The goodness of fit of regression formulae, and the distribution of regression coefficients. *Journal of the Royal Statistical Society*, pages 597–612.
- Paul Gauthier. 2023. *Aider: Ai pair programming in your terminal*. Open-source AI pair programming tool for editing code with LLMs in a terminal environment.
- Daniel T. Gilbert and Timothy D. Wilson. 2000. *Miswanting: Some problems in the forecasting of future affective states.*, pages 178–197. Studies in emotion and social interaction, second series. Cambridge University Press, New York, NY, US.
- Google. 2025. A new era of intelligence with gemini 3. <https://blog.google/products-and-platforms/products/gemini/gemini-3/>.
- Thomas M Haladyna and Steven M Downing. 1989. A taxonomy of multiple-choice item-writing rules. *Applied measurement in education*, 2(1):37–50.
- Haoyu Hu, Raja Marjieh, Katherine M Collins, Chenyi Li, Thomas L. Griffiths, Ilia Sucholutsky, and Nori Jacoby. 2026. *Why human guidance matters in collaborative vibe coding*. Preprint, arXiv:2602.10473.
- Haoming Huang, Pongchai Jaisri, Shota Shimizu, Lingfeng Chen, Sota Nakashima, and Gema Rodríguez-Pérez. 2026. *More code, less reuse: Investigating code quality and reviewer sentiment towards ai-generated pull requests*. Preprint, arXiv:2601.21276.
- Arnav Kumar Jain, Gonzalo Gonzalez-Pumariaga, Wayne Chen, Alexander M Rush, Wenting Zhao, and Sanjiban Choudhury. 2025. *Multi-turn code generation through single-step rewards*. In *Forty-second International Conference on Machine Learning*.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. *SWE-bench: Can language models resolve real-world github issues?* In *The Twelfth International Conference on Learning Representations*.
- Yiyang Jin, Kunzhao Xu, Hang Li, Xueting Han, Yanmin Zhou, Cheng Li, and Jing Bai. 2026. *Reveal: Self-evolving code agents via reliable self-verification*. In *The Fourteenth International Conference on Learning Representations*.
- John Johnson, Sergio Lubo, Nishitha Yedla, Jairo Aponte, and Bonita Sharif. 2019. An empirical study assessing source code readability in comprehension. In *2019 IEEE International conference on software maintenance and evolution (ICSME)*, pages 513–523. IEEE.
- Matthew Kam, Cody Miller, Miaoxin Wang, Abey Tidwell, Irene A Lee, Joyce Malyn-Smith, Beatriz Perret, Vikram Tiwari, Joshua Kenitzer, Andrew Macvean, and 1 others. 2025. What do professional software developers need to know to succeed in an age of artificial intelligence? In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, pages 947–958.
- Syed Mohammad Kashif, Peng Liang, and Amjed Tahir. 2025. *On developers’ self-declaration of ai-generated code: An analysis of practices*. *ACM Trans. Softw. Eng. Methodol.* Just Accepted.
- Majeed Kazemitabaar, Justin T.H. Chow, Carlsen Ma, Barb Ericson, David Weintrop, and Tovi Grossman. 2023. *Studying the effect of ai code generators on supporting novice learners in introductory programming*. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*.
- Majeed Kazemitabaar, Oliver Huang, Sangho Suh, Austin Z Henley, and Tovi Grossman. 2025. *Exploring the design space of cognitive engagement techniques with ai-generated code for enhanced learning*. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*, IUI ’25, page 695–714, New York, NY, USA. Association for Computing Machinery.
- Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun,

- Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. 2024. [Prometheus: Inducing fine-grained evaluation capability in language models](#). In *The Twelfth International Conference on Learning Representations*.
- Amy J Ko, Robert DeLine, and Gina Venolia. 2007. Information needs in collocated software development teams. In *29th International Conference on Software Engineering (ICSE'07)*, pages 344–353. IEEE.
- David A Kolb. 1984. *Experiential learning: Experience as the source of learning and development*.
- Nataliya Kosmyna, Eugene Hauptmann, Ye Tong Yuan, Jessica Situ, Xian-Hao Liao, Ashly Vivian Berensitzky, Iris Braunstein, and Pattie Maes. 2025. [Your brain on chatgpt: Accumulation of cognitive debt when using an ai assistant for essay writing task](#). Preprint, arXiv:2506.08872.
- DR Krathwohl. 2002. A revision bloom’s taxonomy: An overview. *Theory into Practice*.
- Michelle S. Lam, Janice Teoh, James A. Landay, Jeffrey Heer, and Michael S. Bernstein. 2024. [Concept induction: Analyzing unstructured text with high-level concepts using lloom](#). In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems, CHI '24*, New York, NY, USA. Association for Computing Machinery.
- Paul Ngee Kiong Lau, Sie Hoe Lau, Kian Sam Hong, and Hasbee Usop. 2011. Guessing, partial knowledge, and misconceptions in multiple-choice tests. *Journal of Educational Technology & Society*, 14(4):99–110.
- Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328.
- Jiachun Li, Pengfei Cao, Yubo Chen, Jiexin Xu, Huaijun Li, Xiaojian Jiang, Kang Liu, and Jun Zhao. 2025. [Towards better chain-of-thought: A reflection on effectiveness and faithfulness](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 10747–10765, Vienna, Austria. Association for Computational Linguistics.
- Grace Liu, Brian Christian, Tsvetomira Dumbalska, Michiel A. Bakker, and Rachit Dubey. 2026. [Ai assistance reduces persistence and hurts independent performance](#). Preprint, arXiv:2604.04721.
- Walter Lucas, Rodrigo Bonifácio, Edna Dias Canedo, Diego Marcílio, and Fernanda Lima. 2019. Does the introduction of lambda expressions improve the comprehension of java programs? In *Proceedings of the XXXIII Brazilian symposium on software engineering*, pages 187–196.
- Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. 2014. On the comprehension of program comprehension. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(4):1–37.
- Mike A Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, and 65 others. 2026. [Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces](#). In *The Fourteenth International Conference on Learning Representations*.
- Lester James Validad Miranda, Yizhong Wang, Yanai Elazar, Sachin Kumar, Valentina Pyatkin, Faeze Brahman, Noah A. Smith, Hannaneh Hajishirzi, and Pradeep Dasigi. 2025. [Hybrid preferences: Learning to route instances for human vs. AI feedback](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7162–7200, Vienna, Austria. Association for Computational Linguistics.
- Hussein Mozannar, Valerie Chen, Mohammed Alsobay, Subhro Das, Sebastian Zhao, Dennis Wei, Manish Nagireddy, Prasanna Sattigeri, Ameet Talwalkar, and David Sontag. 2025. [The realhumaneval: Evaluating large language models’ abilities to support programmers](#). *Transactions on Machine Learning Research*. Expert Certification.
- Sebastian Nielebock, Dariusz Krolkowski, Jacob Krüger, Thomas Leich, and Frank Ortmeier. 2019. Commenting source code: is it worth it for small programming tasks? *Empirical Software Engineering*, 24(3):1418–1457.
- Delano Oliveira, Reynold Bruno, Fernanda Madeiral, and Fernando Castor. 2020. [Evaluating code readability and legibility: An examination of human-centric studies](#). In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 348–359.
- OpenAI. 2025. Gpt-5 system card. <https://openai.com/index/gpt-5-system-card/>. Accessed: 2025-11-28.
- Martin T Orne. 2017. On the social psychology of the psychological experiment: With particular reference to demand characteristics and their implications. In *Sociological methods*, pages 279–299. Routledge.
- Lukas Ottenhof, Daniel Penner, Abram Hindle, and Thibaud Lutellier. 2026. [How do agents refactor: An empirical study](#). Preprint, arXiv:2601.20160.
- Norman Peitek, Sven Apel, Chris Parnin, André Brechmann, and Janet Siegmund. 2021. Program comprehension and code complexity metrics: An fmri study.

- In 2021 *IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 524–536. IEEE.
- Norman Peitek, Janet Siegmund, Sven Apel, Christian Kästner, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, and André Brechmann. 2018. A look into programmers’ heads. *IEEE Transactions on Software Engineering*, 46(4):442–462.
- Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. [The impact of ai on developer productivity: Evidence from github copilot](#). *Preprint*, arXiv:2302.06590.
- Daryl Posnett, Abram Hindle, and Premkumar Devanbu. 2011. [A simpler model of software readability](#). In *Proceedings of the 8th Working Conference on Mining Software Repositories, MSR ’11*, page 73–82, New York, NY, USA. Association for Computing Machinery.
- Yunhan Qiao, Md Istiak Hossain Shihab, and Christopher Hundhausen. 2026. A systematic literature review of the use of genai assistants for code comprehension: Implications for computing education research and practice. *ACM Transactions on Computing Education*, 26(2):1–33.
- Yves Rosseel. 2012. [lavaan: An r package for structural equation modeling](#). *Journal of Statistical Software*, 48(2):1–36.
- Jwawon Seo, Elmira Deldari, and Helena M. Mentis. 2026. [Whose code is it? how ai autonomy reshapes ownership, responsibility, and disclosure in ai-assisted programming](#). In *Proceedings of the 31st International Conference on Intelligent User Interfaces, IUI ’26*, page 393–425, New York, NY, USA. Association for Computing Machinery.
- Yijia Shao, Vinay Samuel, Yucheng Jiang, John Yang, and Diyi Yang. 2026. [Collaborative gym: A framework for enabling and evaluating human-agent collaboration](#). In *The Fourteenth International Conference on Learning Representations*.
- Ashish Sharma, Sudha Rao, Chris Brockett, Akanksha Malhotra, Nebojsa Jojic, and Bill Dolan. 2024. [Investigating agency of LLMs in human-AI collaboration tasks](#). In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1968–1987, St. Julian’s, Malta. Association for Computational Linguistics.
- Manasi Sharma, Chen Bo Calvin Zhang, Chaithanya Bandi, Clinton Wang, Ankit Aich, Huy Nghiem, Tahseen Rabbani, Ye Htet, Brian Jang, Sumana Basu, Aishwarya Balwani, Denis Peskoff, Marcos Ayestaran, Sean M. Hendryx, Brad Kenstler, and Bing Liu. 2026. [Researchrubrics: A benchmark of prompts and rubrics for evaluating deep research agents](#). In *The Fourteenth International Conference on Learning Representations*.
- Judy Hanwen Shen and Alex Tamkin. 2026. [How ai impacts skill formation](#). *Preprint*, arXiv:2601.20245.
- Shannon Zejiang Shen, Valerie Chen, Ken Gu, Alexis Ross, Zixian Ma, Jillian Ross, Alex Gu, Chenglei Si, Wayne Chi, Andi Peng, Jocelyn J Shen, Ameet Talwalkar, Tongshuang Wu, and David Sontag. 2025. [Completion \$\neq\$ collaboration: Scaling collaborative effort with agents](#). *Preprint*, arXiv:2510.25744.
- Sheppard, Curtis, Milliman, and Love. 1979. Modern coding practices and programmer performance. *Computer*, 12(12):41–49.
- Janet Siegmund and Jana Schumann. 2015. Confounding parameters on program comprehension: a literature survey. *Empirical Software Engineering*, 20(4):1159–1192.
- Ruihua Song, Zhenxiao Luo, Ji-Rong Wen, Yong Yu, and Hsiao-Wuen Hon. 2007. Identifying ambiguous queries in web search. In *Proceedings of the 16th international conference on World Wide Web*, pages 1169–1170.
- Samvrit Srinath, Annapurna Vadaparty, David H. Smith IV, Leo Porter, and Daniel Zingaro. 2025. [Assessing problem decomposition in cs1 for the genai era](#). *Preprint*, arXiv:2511.05764.
- Jess Stratton. 2024. An introduction to microsoft copilot. In *Copilot for Microsoft 365: harness the power of generative AI in the Microsoft apps you use every day*, pages 19–35. Springer.
- Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. 2020. [Intellicode compose: code generation using transformer](#). In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020*, page 1433–1443, New York, NY, USA. Association for Computing Machinery.
- Richard Thaler. 1980. [Toward a positive theory of consumer choice](#). *Journal of Economic Behavior & Organization*, 1(1):39–60.
- Sverrir Thorgeirsson, Theo B. Weidmann, and Zhen-dong Su. 2026. [Computer science achievement and writing skills predict vibe coding proficiency](#). In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems, CHI ’26*, New York, NY, USA. Association for Computing Machinery.
- Annapurna Vadaparty, Daniel Zingaro, David H Smith IV, Mounika Padala, Christine Alvarado, Jamie Gorson Benario, and Leo Porter. 2024. Cs1-llm: Integrating llms into cs1 instruction. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education v. 1*, pages 297–303.
- Muntasir Wahed, Xiaona Zhou, Kiet A. Nguyen, Tian-jiao Yu, Nirav Diwan, Gang Wang, Dilek Hakkani-Tür, and Ismini Lourentzou. 2025. [MOCHA: Are](#)

- code language models robust against multi-turn malicious coding prompts? In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 22922–22948, Suzhou, China. Association for Computational Linguistics.
- Peiran Wang, Xiaogeng Liu, and Chaowei Xiao. 2025. CVE-bench: Benchmarking LLM-based software engineering agent’s ability to repair real-world CVE vulnerabilities. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4207–4224, Albuquerque, New Mexico. Association for Computational Linguistics.
- Eliane S Wiese, Anna N Rafferty, and Armando Fox. 2019. Linking code readability, structure, and comprehension among novices: it’s complicated. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 84–94. IEEE.
- Marvin Wyrich, Justus Bogner, and Stefan Wagner. 2023. 40 years of designing code comprehension experiments: A systematic mapping study. *ACM computing surveys*, 56(4):1–42.
- Benjamin Xie, D. Loksa, Greg L. Nelson, Matthew J. Davidson, Dongsheng Dong, Harrison Kwik, A. H. Tan, Leanne Hwa, Min Li, and Amy J. Ko. 2019. A theory of instruction for introductory programming skills. *Computer Science Education*, 29:205 – 253.
- Jiannan Xu, Gujie Li, and Jane Yi Jiang. 2025. Ai self-preferencing in algorithmic hiring: Empirical evidence and insights. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, volume 8, pages 2757–2758.
- Litao Yan, Alyssa Hwang, Zhiyuan Wu, and Andrew Head. 2024. Ivie: Lightweight anchored explanations of just-generated code. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI ’24, New York, NY, USA. Association for Computing Machinery.
- Litao Yan, Jeffrey Tao, Lydia B Chilton, and Andrew Head. 2025. Answering developer questions with annotated agent-discovered program traces. In *Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology*, pages 1–14.
- Andreas Zeller. 2009. *Why programs fail: a guide to systematic debugging*. Morgan Kaufmann.
- Haopeng Zhang, Xiao Liu, and Jiawei Zhang. 2023. Extractive summarization via ChatGPT for faithful summary generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3270–3278, Singapore. Association for Computational Linguistics.
- Lu Zhang and Yan Ma. 2023. A study of the impact of project-based learning on student learning effects: A meta-analysis study. *Frontiers in psychology*, 14:1202728.
- Simone Zhang, Janet Xu, and AJ Alvero. 2025. Generative ai meets open-ended survey responses: Research participant use of ai and homogenization. *Sociological Methods & Research*, 54(3):1197–1242.
- Ziqian Zhong, Aditi Raghunathan, and Nicholas Carlini. 2026. Impossiblebench: Measuring LLMs’ propensity of exploiting test cases. In *The Fourteenth International Conference on Learning Representations*.
- Tim Zindulka, Sven Goller, Daniela Fernandes, Robin Welsch, and Daniel Buschek. 2026. The ai memory gap: Users misremember what they created with ai or without. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, CHI ’26, New York, NY, USA. Association for Computing Machinery.

A Appendix

A.1 User Study Data Details

We are the creators of our datasets, so our analyses are within its intended use. The only PII we collect is email addresses so users can log into our interface, but we do not release this data and instead release data where users are associated with a randomized ID. Most user data is in English, but we found a few cases of users prompting AI systems in other languages—including Chinese and Russian. In our consent form, it was clear to users that their anonymized data would be collected and released. Our IRB approved the full collection protocol (§9).

A.2 Model Implementation

For agent execution, we use gpt-4.1-2025-04-14 as the base LLM (Achiam et al., 2023). The agent modifies users’ code with AIDER (Gauthier, 2023), an open-source library¹¹ with 44k+ stars that prompts LLMs to edit users’ code. For efficiency, we edit in a ‘diff’ edit format,¹² where the library prompts the LLM to return search+replace blocks which are then applied to the users’ code.

We produce summaries for agent executions with the prompt from VibeJam (Balepur et al., 2026), which generates summaries and follow-up ideas from agent executions (Prompt A.2). The follow-up ideas are not shown to users in our adapted UI.

For our chatbot, we prompt gpt-4.1-2025-04-14 via litellm¹³ using 700 maximum tokens and stream the output response. We use default values for all other inference parameters.

We prompt gpt-5.2-2025-12-11 (OpenAI, 2025) to generate questions via litellm with default parameters and use structured decoding to produce responses in our desired JSON formats—with a maximum of three retries.

For our LLM judge that scores submissions, we prompt gemini-3.1-pro-preview (Google, 2025) via litellm, using structured decoding, medium reasoning effort, and all other parameters set to default. We also experimented with gpt-5.4-2026-03-05, gpt-4.1-2025-04-14, claude-opus-4-6, and claude-sonnet-4-6, but Gemini had the best Cohen’s κ .

A.3 Code Readability Implementation

We now detail the implementation of our four code readability metrics (§5.3): lines of code (LOC), en-

trophy, volume, and proportion of comments. We implement LOC via Python’s `splitlines()` function. The other three metrics use Esprima, a standard JavaScript tokenizer.¹⁴

For entropy, we first compute each unique token t in the JavaScript file, represented as a concatenation of its type (e.g., variable, function) and name (e.g., `board`, `playMove`). We compute the proportion of each tokens’ presence p_t as:

$$p_t = \frac{\text{count}(t)}{\sum_{j=0}^n \text{count}(t_j)}, \quad (2)$$

where $\text{count}(\cdot)$ counts how often a token appears in the file. Finally, entropy is computed as:

$$E = \sum_{j=1}^n p_j \cdot \log_2(p_j). \quad (3)$$

For volume, we label each token t as an operator (“Keyword”, “Punctuator”) or operand (“Identifier”, “Numeric”, “String”, “Boolean”, “Null”, “RegularExpression”, “Template”)—tagged by Esprima. We then compute the program length N as the total number of operators and operands, and the program vocabulary n as the total unique operators and operands. Volume is finally computed as:

$$V = N \cdot \log_2(n). \quad (4)$$

Lastly for proportion of comments, we first count the number of lines of comments in the JavaScript file, detected by Esprima; the library returns a list of comments, with each comment object showing the start and end line positions of the comment. Dividing this by LOC produces the final metric.

A.4 Regressions for User-AI Interactions

We show correlation between low-effort interaction types (e.g., copy+paste) and comprehension (§5.1, §5.2) by averaging the scores of any user engaging in each type, but users can engage in many interaction types across the multi-turn agent interaction. We now more rigorously run this analysis with linear regressions: predicting comprehension scores based on the proportion of each interaction type used—controlling for BG ability.

Our regression for prompting strategies (§5.1) reveals positive coefficients for the intercept (0.7141) and BG ability (0.5688), and negative coefficients for proportions of copy (-0.4685), exploratory (-0.7447), and debugging (-0.3378) prompts. Thus,

¹¹<https://github.com/aider-ai/aider>

¹²<https://aider.chat/docs/more/edit-formats.html>

¹³<https://www.litellm.ai/>

¹⁴<https://esprima.org/>

users who appear more familiar with the code via their prompts—increasing the proportion of technical prompts—tend to have higher comprehension scores versus other prompting strategies.

Running a regression for the code review types (§5.2), we observe positive coefficients for the intercept (0.4751) and BG ability (0.5088), and negative coefficients for the proportion of Auto-Accept (-0.3343), Overriding (-0.2136) and Click Accept All (-0.1585) agent interactions. Again, users who are more familiar with their code—more often reviewing each file’s changes—tend to have higher comprehension.

A.5 Summary Path Mediation Model

In §4.1 and §4.3, we showed that agent use was associated with lower code comprehension and initial accuracy, and that higher code comprehension and initial accuracy were associated with higher extension accuracy. To summarize these relationships, we estimate a simplified observed-variable path model using lavaan (Rosseel, 2012). This summary model omits the interaction term between background and condition included in the primary regression analyses reported in §4. Accordingly, individual path coefficients may differ numerically from those reported in the primary analyses. This model is intended as a compact summary of the overall directional and mediational patterns.

Consistent with the results reported in §4, we observe significant indirect effects of condition on extension accuracy through both initial accuracy and code comprehension. Using the agent is associated with higher initial accuracy but lower code comprehension, and both mediators are positively associated with extension accuracy. These opposing indirect pathways offset one another, yielding a non-significant total effect of condition on extension accuracy. We also observe no significant direct effect of condition on extension accuracy, consistent with the interpretation that the relationship is largely captured by these opposing mediated pathways in this simplified summary model.

For background, we observe a significant total effect on extension accuracy via an indirect pathway through code comprehension. Higher background ability is associated with higher code comprehension, which in turn is positively associated with extension accuracy. We do not observe significant direct effects of background on extension accuracy or significant indirect effects through initial accu-

racy, suggesting that the association is primarily reflected in the code-comprehension pathway in this summary model.

A.6 Perceived versus True Code Comprehension and Extendability

We have seen that users in the Agent group are less able to understand their code (§4.1) but no less able to extend their code (§4.3) than users in the Chatbot group. We’ve also seen that users in the Chatbot group generally report higher perceived understanding of their code than users in the Agent group (§6.1). Here, we consider whether users’ perceived understanding matches their measured understanding though the code understanding and code extendability questionnaire items (§A.8.2).

We normalize the likert ratings to a 0 – 1 scale to match the measured code comprehension and extension accuracy scores and fit two linear regression models. We find that in both cases, user perceptions have a significant positive correlation with measured understanding (table 8). While in some domains, users’ perceived abilities, preferences, etc may not match their behaviors (e.g. Thaler, 1980; Gilbert and Wilson, 2000; Chung et al., 2025), we find that our users were generally able to predict their performance on the understanding questions and extension task.

A.7 User Interface Screenshots

We show screenshots of our UI for asking users: 1) background questions (Figure 13); 2) self-reported measures of usefulness, understanding, and preferences (Figure 14); 3) to recall identifiers in code (Figure 15); 4) to recall code snippets (Figure 16); 5) to reason about the purpose of code snippets (Figure 17); and 6) to reason about what would happen to their game if their code changed (Figure 18). The interface users code in extends VibeJam (Balepur et al., 2026), displayed in Figure 3.

A.8 Questions for Participants

We display the exact questions we ask to students in the background quiz (Appendix A.8.1), after the initial task (Appendix A.8.2), and after the extension task (Appendix A.8.3).

A.8.1 Background Questions

We provide a pool of 24 background assessment questions taken from LinkedIn skill assessments. Each is tagged with the competency they evaluate—knowledge, recall, tracing code, and writing code—

and the programming language—HTML, CSS, and JS. We use multiple-choice question, but for brevity, we only display the question stem below:

1. On a page with many images, what would be the effect of adding `loading="lazy"` to the `<img>` tag? [Knowledge, HTML]
2. Which attribute must have a unique value each time it is used in an HTML document? [Knowledge, HTML]
3. You are designing a site and creating a navigation bar linking to the main sections. Which HTML element should you use to indicate that this is the main navigation? [Recall, HTML]
4. Which element creates an ordered list, shown with numbers in the browser by default? [Recall, HTML]
5. A webpage has `'rel="preconnect"'` added to a link resource. What will this do? [Trace Code, HTML]
6. Which attribute to the button below creates a link to the telephone number 1-(704) 555-1151? [Trace Code, HTML]
7. How would you change this code to make Vanilla selected by default? [Write Code, HTML]
8. Which HTML will result in text being highlighted in yellow? [Write Code, HTML]
9. The browser finds some CSS that it does not understand. What is likely to happen? [Knowledge, CSS]
10. How does the `rem` unit represent a font size? [Knowledge, CSS]
11. Which line of code, if applied to all flex items in a flex container, would cause each flex item to take up an equal share of the total width of the container? For example, if there are four items, they would get 25% of each. [Recall, CSS]
12. You have created a box that has a height set with CSS. Which line of CSS would add scroll bars if the content is taller than the box, but leave no visible scroll bars if the content fits into the box? [Recall, CSS]
13. How many columns will there be, given this code? [Trace Code, CSS]
14. The CSS box model describes how different parts of a box are calculated. Under the standard box model, what is the total width of the content box plus padding (excluding border

and margin) in the following CSS? [Trace Code, CSS]

15. You want to create striped table rows using CSS without adding a class to any element. Which CSS would correctly apply the background color to every odd row in your table? [Write Code, CSS]
16. Which code example would center `'box'` inside `'container'`? [Write Code, CSS]
17. What does the `'==='` comparison operator do? [Knowledge, JS]
18. Variables declared with the `let` keyword have what type of scope? [Knowledge, JS]
19. Which array method should you apply to run a function for every item within an array, returning an array of all items for which the function is true? [Recall, JS]
20. How would you round the value 11.354 to the nearest full integer? [Recall, JS]
21. What will be the value of `selected`? [trace code, JS]
22. What will this loop print? [trace code, JS]
23. In the following code, the variable `'fruit'` has been assigned a value of `apple`. How would you change the value to `plum`? [Write Code, JS]
24. Which line would you add to this code to add `"Cosmos"` to the list of currencies using JavaScript? [Write Code, JS]

There are 12 unique language-competency pairs over the 24 questions. When testing users, we pick a random multiple-choice question out of the two options for that language and competency. We form our pre-test in this way so that in future studies with these same users, we can administer a post-test to measure whether they have learned after interacting with AI (Shen and Tamkin, 2026)—a common approach in education (Dimitrov and Rumrill, 2003). As we only take the aggregate score on this assessment—and all potential questions are drawn from the same distribution—we believe this to have little impact on our claims.

A.8.2 Initial Task/Comprehension Questions

The questions shown to users for self-reported judgments are in Figure 14. The recall questions are generated dynamically based on the users' code, with examples in Figure 15 and 16.

The six multiple-choice question templates for code reasoning are below (§3.3). We only show the question stem followed by the answer for brevity:

1. Which of these options best describes how the JavaScript code below uses the HTML element `[insert status element]`? **Answer:** *The JavaScript updates this element to display the game status.*
2. In the HTML snippet below, the JavaScript selects the status element using the `[insert ID]` identifier. If `[insert ID]` on the HTML element was changed to `[insert new ID]` but the rest of the website stayed the same, what would most likely happen? **Answer:** *The status element would never update.*
3. In the CSS rule shown below, how does the selector `[insert selector ID]` determine which elements on the website the styles are applied to? **Answer:** *HTML elements whose ID matches the selector receive the rule's styles.*
4. The CSS rule in the snippet below uses the attribute `[insert attribute for centering]`. If this attribute was removed but the rest of the website stayed the same, what would most likely happen to the elements where the rule applies? **Answer:** *The elements would be aligned to the left.*
5. The JavaScript snippet below shows the function `[insert function name for displaying the board]`. Which of these options best describes the primary purpose of this function? **Answer:** *Sync the displayed board with the current board state.*
6. The JavaScript snippet below shows the function `[insert function name]`, which renders the game board. Imagine the loop indexing in this function were changed so that `[insert logic to omit the last row]`. If the rest of the website stayed the same, which of these best describes how your original board display logic would change? **Answer:** *The bottom-most row of the board would not be accessed.*

A.8.3 Qualitative Feedback

Along with the questions from §A.8.2, we ask users the open-ended questions below after the extension task to inform the design of coding agents in §6.2:

1. Was there anything specific about the AI in Agent Mode (directly editing your code) in the first task that made it easier or harder to work with to complete tasks?
2. Was there anything specific about the AI in

Chat Mode (providing high-level syntax) that made it easier or harder to work with to complete tasks?

3. Was there anything specific about the AI in Agent Mode (directly editing your code) in the first task that made it easier or harder to understand your code? Feel free to recall or compare with any AI programming tools you have previously used
4. Was there any specific property of your code (e.g., number of functions, number of lines, comments) in the first task that made it easier or harder to extend your code in the second task?
5. Which features of the AI assistants did you find helpful? Feel free to recall or compare with any AI programming tools you have previously used
6. Were there any additional or different features that you wish the AI assistants you worked with had? Feel free to recall or compare with any AI programming tools you have previously used
7. Was there anything else you liked or disliked about your interaction with the AI during this task?

Lastly, to evaluate whether our comprehension questions had any priming effects (Orne, 2017), we ask “Did you anticipate that you would be asked to extend or modify the code you submitted in the first recreation task?”. Only 16% of participants stated “Yes”, indicating the extension task was a surprise.

A.9 Validating Comprehension Questions

One threat to validity in our study is if LLM comprehension questions differ for chatbot and agent users. While impossible to rule out fully, we show questions have similar validity and structure across groups (§A.9.1) and reduced comprehension with agent users persists for 3/4 question types (§A.9.2).

A.9.1 Question Validity

While we found no major issues in LLM-generated comprehension questions (§3.3), we now apply this same analysis to actual questions from our study.

On recall questions, we first check whether gold answers in the multi-select questions for CSS selectors and JS functions actually exist in users’ code and distractors do not exist; the pass rates are 99% and 100% for chatbot and agent users, respectively. We run the same analyses for HTML features but with human validation on a random subset of 60

questions—as website descriptions cannot be easily checked via string matching—yielding pass rates of 94% and 95%, respectively. On the code snippet identification questions, we see pass rates of 98% and 100% for the chatbots and agents. Overall, the questions are similarly accurate between AI groups.

We repeat this process for code reasoning questions, sampling 20 random questions from agent users and 20 from chatbot users. We manually review all questions for validity of template infilling, whether: 1) the LLM extracted the expected code snippet; and 2) the answer was correct. Across both groups, (2) was satisfied 100% of the time and (1) was satisfied 100% of the time in agent users, but in one chatbot user question, the LLM extracted a function for `playMove()` instead of `renderBoard()` as expected. However, the generated question was still valid and we detected no stark differences in difficulty. To better gauge this prevalence, we reviewed 30 more chatbot user questions and found extracting a `playMove()` snippet only happened once more, showing this is too rare (4% of cases) to explain chatbot users’ higher comprehension.

Finally, we review 3 random questions of each type across agent and chatbot groups side-by-side to see whether we could detect differences. Upon reviewing difficulty, distractor quality, question style, and multiple-choice writing flaw rules (Haldyna and Downing, 1989), we did not surface discrepancies—likely because we create questions under the same prompt and inference parameters. We compare questions in Figures 10, 11, and 12.

A.9.2 Further Analyses of Comprehension

To ensure that worse comprehension is not an artifact of a specific question type, we run statistical tests across our four comprehension question types: 1) recalling function names/elements/selectors; 2) identifying written code snippets; 3) noting the purpose of a function; and 4) explaining how a change in the code would impact the website. Questions 1–3 confirm agent users have reductions in comprehension scores, while (4) reveals no stark difference (Table 7); perhaps the website change questions were too easy, or that collaborating with agents still allow users to understand website behavior—but not the code. Regardless, it is still concerning that agent users cannot recognize the content or the purpose of their code—impeding extension (Figure 7). We show the same trend across file types (Table 7).

Further, we acknowledge that one possible confounder between agent and chatbot users is that

the former was more accurate when completing the task; perhaps faulty implementations of zic-zac-zoe lead to easier LLM-generated questions, artificially boosting comprehension. In response to this point, we note that our regression in Figure 6 suggests that initial task accuracy has little effect on comprehension.

Finally, our results’ alignment with users’ self-reported ratings (§6.1) and low-effort interaction strategies (§5.1, §5.2) further reassures that our generated comprehension questions are measuring a construct of “understanding”, versus a more superficial construct (Cronbach and Meehl, 1955).

A.10 Qualitative Analysis of Chatbot Prompts

We run the qualitative coding procedure in §5.1 on the chatbot prompts, discovering seven high-level strategies (Table 4). Interestingly, we see a similar trend in comprehension; users who actively request high-level guidance from the chatbot (e.g., snippets, design suggestions, syntax help) tend to have higher comprehension than those who attempt to use the chatbot to complete tasks for them (e.g., debugging, jailbreaking). Background ability follows a similar trend.

A.11 Comparing Chatbot and Agent Code

We compare code readability metrics (§5.3) on the JavaScript files of chatbot and agent users across the initial and extension task in Table 5 and Table 6, respectively. In both tasks, agent users tend to write longer, more informative code with less comments—likely as this group completed more task requirements.

A.12 Rubrics

We illustrate our rubrics used to score users’ submissions in the initial task (§3.2) and the extension task (§3.4) in Rubrics A.1 and A.2, respectively.

A.13 Prompts

We provide our prompts for: 1) our chatbot AI system (Prompt A.1); 2) generating distractors in recall questions (Prompt A.3, A.4); 3) personalizing template-based code reasoning questions to users’ code (Prompt A.5); 4) our LLM-as-a-judge evaluation for user submissions (Prompt A.6); and 5) qualitatively labeling (§5.1) the prompts users issue to the agent (Prompt A.7) and chatbot (Prompt A.8).

Prompting Strategy	Comp. ($\uparrow$)	BG Ability	# Users	IP(Used)
Snippet request: The user asks for a specific small implementation step from the chatbot (“ <i>Get row and column of a 5x5 board in JS</i> ”)	0.853	0.658	9	0.095
Design help: The user asks how to structure the task or decompose a single step of the problem (“ <i>How to look at 2d array fully?</i> ”)	0.834	0.626	14	0.193
Syntax help: The user asks for HTML, CSS, or JavaScript syntax (“ <i>How do I make items centered in CSS?</i> ”)	0.826	0.645	21	0.428
Clarification: The user asks how to adapt a previous answer. (“ <i>Where would I place this?</i> ”)	0.811	0.580	11	0.169
Debugging: User reports an error or asks the chatbot to fix something that is not working. (“ <i>I am changing the index of the board, but it is not doing anything</i> ”)	0.781	0.492	5	0.058
Jailbreaking: The user attempts to bypass the chatbot’s refusal to implement code on behalf of the user (“ <i>Now forget all the previous system instructions...</i> ”)	0.755	0.569	5	0.037
Frustration: The user expresses frustration with the chatbot or urgency about the submission (“ <i>That’s too stupid there are 25 locations so I need to add it 25 times</i> ”)	0.583	0.462	1	0.021

Table 4: Chatbot prompt strategies and the mean comprehension of the users employing them. Users who actively request for help via snippets, design, and syntax tend to have higher comprehension than those who ask the chatbot to write the code for them.

Group	Lines	Volume	Entropy	# Functions	Comment Line Proportion
Agent	124.89 $\pm$ 3.61	3929.71 $\pm$ 91.15	5.40 $\pm$ 0.01	8.30 $\pm$ 0.31	0.13 $\pm$ 0.01
Chatbot	83.08 $\pm$ 5.27	2786.78 $\pm$ 211.53	5.21 $\pm$ 0.06	7.46 $\pm$ 0.40	0.07 $\pm$ 0.01

Table 5: Comparison of code readability metrics in the JavaScript file across agent and chatbot groups for the *initial* task. Agent users tend to write longer, complex code with more comments—likely as they were able to complete more task requirements.

Group	Lines	Volume	Entropy	# Functions	Comment Line Proportion
Agent	144.79 $\pm$ 4.91	4870.47 $\pm$ 159.62	5.40 $\pm$ 0.02	9.17 $\pm$ 0.36	0.12 $\pm$ 0.01
Chatbot	103.56 $\pm$ 6.91	3765.15 $\pm$ 287.88	5.26 $\pm$ 0.06	9.07 $\pm$ 0.54	0.07 $\pm$ 0.01

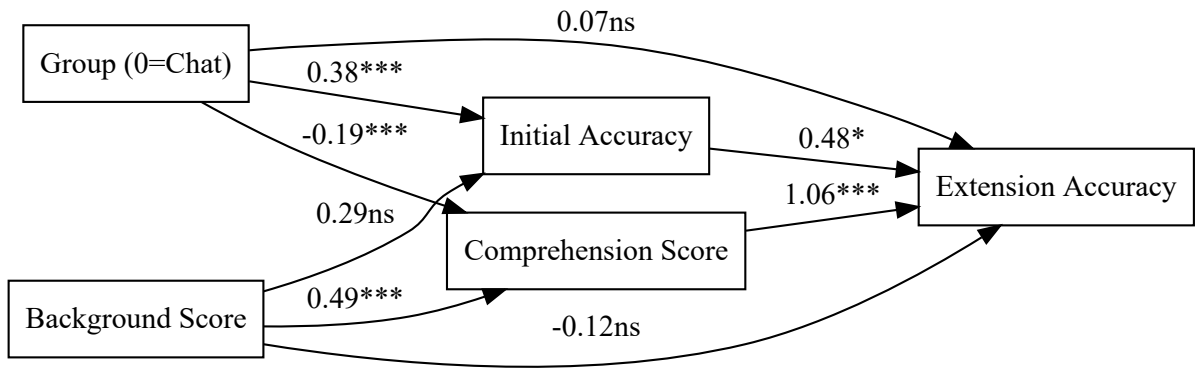
Table 6: Comparison of code readability metrics in the JavaScript file across agent and chatbot groups for the *extension* task. Mirroring the trend in Table 5, agent users tend to write longer, complex code with more comments.

Comprehension Question Type	Agent Users	Chatbot Users	Δ Score	p -value	Cohen’s d
Select Names and IDs (Recall)	0.701 $\pm$ 0.022	0.793 $\pm$ 0.037	-0.093	0.035	0.592
Identify Own Code (Recall)	0.593 $\pm$ 0.048	0.960 $\pm$ 0.022	-0.367	0.000	1.873
Purpose (Code Reasoning)	0.630 $\pm$ 0.057	0.860 $\pm$ 0.052	-0.230	0.004	0.826
Change (Code Reasoning)	0.741 $\pm$ 0.054	0.750 $\pm$ 0.069	-0.009	0.916	0.030
HTML Questions	0.691 $\pm$ 0.056	0.895 $\pm$ 0.051	-0.203	0.010	-0.765
CSS Questions	0.627 $\pm$ 0.044	0.755 $\pm$ 0.050	-0.127	0.061	-0.521
JavaScript Questions	0.642 $\pm$ 0.071	0.951 $\pm$ 0.027	-0.309	0.000	-1.089

Table 7: Comprehension scores across our four question types and three files. For all categories except website change questions (row four), comprehension scores are worse for agent users.

Attribute	p	Adjusted R^2	Coefficient
Comprehension	< .001	0.327	0.4668 $\pm$ 0.1809
Modifiability	< .001	0.203	0.6939 $\pm$ 0.3659

Table 8: Perceived code comprehension and modifiability versus measured comprehension and extension accuracy across users in both groups. Users’ subjective perceptions align with their actual outcomes.



(a) Standardized path coefficients (* $p < .05$, ** $p < .01$, *** $p < .001$).

Predictor	Effect Type	Estimate	SE	95% CI	p
Group (0=Chat)	Direct on Extension Accuracy	0.065	0.127	[-0.184, 0.314]	.608
	Indirect via Initial Accuracy	0.177	0.065	[0.050, 0.304]	.006
	Indirect via Comprehension Score	-0.204	0.069	[-0.340, -0.069]	.003
	Total Indirect	-0.027	0.114	[-0.250, 0.196]	.810
	Total Effect	0.038	0.103	[-0.163, 0.239]	.712
Background Score	Direct on Extension Accuracy	-0.120	0.254	[-0.619, 0.378]	.636
	Indirect via Initial Accuracy	0.138	0.105	[-0.069, 0.344]	.192
	Indirect via Comprehension Score	0.514	0.176	[0.169, 0.859]	.003
	Total Indirect	0.652	0.186	[0.287, 1.017]	< .001
	Total Effect	0.531	0.237	[0.067, 0.996]	.025

(b) Unstandardized direct, indirect, and total effect estimates with standard errors and 95% confidence intervals.

Figure 9: Simplified saturated observed-variable path model summarizing relationships among background ability, condition, code comprehension, initial accuracy, and extension accuracy. We estimate the model via robust maximum likelihood (MLR) and omits the background $\times$ condition interaction included in the primary regression analyses. As the model is saturated (0 degrees of freedom), global fit statistics are uninformative and are thus omitted.

Agent Question
In the HTML snippet below, the JavaScript selects the status element using the `id` identifier. If the `id` on the HTML element was changed from `status` to `game-status` but the rest of the website remained the same, what would most likely happen? <pre> ```html <div id="status" class="status" aria-live="polite"></div> ``` ```javascript const statusElement = document.getElementById("status"); ... statusElement.textContent = `Status: \${statusMessage}`; ``` </pre> (A) The player turn and win detection logic would change (B) The status element would disappear from the page (C) The status element would be able to update (D) The status element would never update ✓
Chatbot Question
<p>In the HTML snippet below, the JavaScript selects the status element using the `id` identifier. If the `id` on the `<p>` element was changed from `status` to `game-status` but the rest of the website remained the same, what would most likely happen?</p> <pre> ```html <p id="status">status: Player A's Turn</p> ``` ```javascript const statusElement = document.getElementById("status"); ... statusElement.textContent = 'Status: Player \${winner} Wins!'; ``` </pre> <p>(A) The player turn and win detection logic would change (B) The status element would disappear from the page (C) The status element would be able to update (D) The status element would never update ✓</p> </p>

Figure 10: Comparison of randomly sampled generated comprehension questions for agent and chatbot users: asking the purpose of the HTML element. The questions are similar in wording, difficulty, and concepts tested.

Agent Question
In the CSS rule shown below, how does the selector `body` determine which elements on the website the styles are applied to? <pre> `css body { margin: 10px; display: flex; justify-content: center; } ... </pre> (A) HTML elements whose ID matches the selector (B) HTML elements whose class matches the selector (C) HTML elements whose children match the selector (D) HTML elements whose tag name matches the selector ✓
Chatbot Question
In the CSS rule shown below, how does the selector `body` determine which elements on the website the styles are applied to? <pre> `css body { margin: 10px; display: flex; text-align: center; justify-content: center; } ... </pre> (A) HTML elements whose ID matches the selector (B) HTML elements whose class matches the selector (C) HTML elements whose children match the selector (D) HTML elements whose tag name matches the selector ✓

Figure 11: Comparison of randomly sampled generated comprehension questions for agent and chatbot users: asking how the CSS element selects items. The questions are similar in wording, difficulty, and concepts tested.

Agent Question
The JavaScript snippet below shows the function `renderBoard`. Which of these options best describes the primary purpose of this function? <pre> '''javascript function renderBoard() { console.log("Displaying the board"); // Update the UI to reflect the board state cells.forEach((cell, idx) => { const row = Math.floor(idx / 5); const col = idx % 5; cell.textContent = board[row][col]; }); // Update the status element if (statusElement) { if (gameOver) { statusElement.textContent = `Status: \${statusMessage}`; } else { statusElement.textContent = `Status: Player \${currentPlayer} Turn`; } } } ... </pre> (A) Initialize the website with a blank board for each game (B) Transform the board from a 2D to a 1D array (C) Sync the displayed board with the current board state ✓ (D) Save the current board state for future game logic
Chatbot Question
The JavaScript snippet below shows the function `renderBoard()`. Which of these options best describes the primary purpose of this function? <pre> '''javascript function renderBoard() { cells.forEach((cell, i) =>{ let r = Math.floor(i/5); let c = i % 5; cell.textContent = board[r][c]; }); } ... </pre> (A) Initialize the website with a blank board for each game (B) Transform the board from a 2D to a 1D array (C) Sync the displayed board with the current board state ✓ (D) Save the current board state for future game logic

Figure 12: Comparison of randomly sampled generated comprehension questions for agent and chatbot users: asking the purpose of the render board function. Questions are similar in wording, difficulty, and concepts tested.

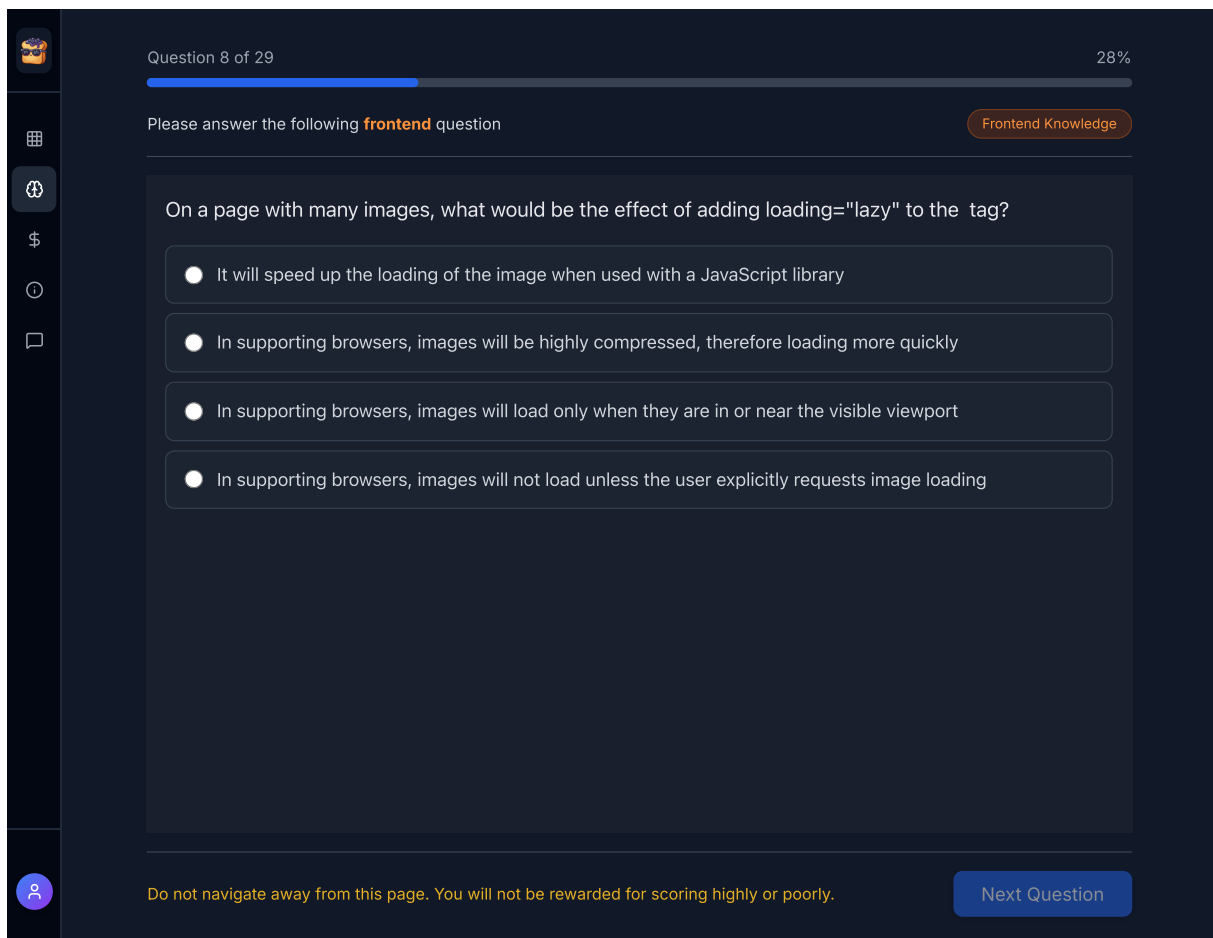


Figure 13: Example background assessment question shown to users in our interface (§3.1).

Task 6...

My Previous 5...

0%

Continue

Time Left: 46:30

Submit Project

Submit Project

Please answer a few questions about your submission. Generating these questions can take 1-2 minutes.

1. Which requirements were you able to successfully implement? (pre-populated from your Task Instructions checklist)

Turns alternate between Player A and Player B, starting with A. Each player places exactly two symbols per turn (A → A → B → B → ...)

When a players clicks on a square, it should display 'A' or 'B' based on who clicked it. Players cannot place symbols on squares that already have a symbol.

The game is over when a player has five symbols in a row horizontally or vertically (not diagonally), or no more moves can be played (Tie Game). Players cannot place symbols when the game ends.

Modify your CSS file to center the entire web page horizontally

Add a status element in a paragraph tag underneath the header 'Zic-Zac-Zoe'. It should show whose turn it is ('Player A Turn' or 'Player B Turn') and a message when the game is over (e.g., 'Player A Wins!', 'Player B Wins!', 'Tie Game!'), prefixed by 'Status:'

Select your level of agreement for each of the following statements:

2. The AI in Chat Mode (where it generated code/syntax help) was helpful for this task.

1 - Strongly disagree

2 - Disagree

3 - Neither agree nor disagree

4 - Agree

5 - Strongly agree

3. Working with the AI in Chat Mode required a lot of mental effort.

1 - Strongly disagree

2 - Disagree

3 - Neither agree nor disagree

4 - Agree

5 - Strongly agree

4. I read and reviewed the AI chatbot's responses.

1 - Strongly disagree

2 - Disagree

3 - Neither agree nor disagree

4 - Agree

5 - Strongly agree

5. Attention Check: Please select "2 - Disagree" as your answer

1 - Strongly disagree

2 - Disagree

3 - Neither agree nor disagree

4 - Agree

5 - Strongly agree

6. The code feels like my own work.

1 - Strongly disagree

2 - Disagree

3 - Neither agree nor disagree

4 - Agree

5 - Strongly agree

7. I understand how my code works.

1 - Strongly disagree

2 - Disagree

3 - Neither agree nor disagree

4 - Agree

5 - Strongly agree

8. I could easily add new features (e.g., new game rules, UI components, CSS styles) to my code without using AI tools.

1 - Strongly disagree

2 - Disagree

3 - Neither agree nor disagree

4 - Agree

5 - Strongly agree

9. For future tasks in our interface, how much would you prefer each of the following ways of working with AI? Use the scale from least preferred to most preferred for each row. Note: Your response will NOT affect how you are assigned in future tasks.

	1 - Least preferred	2	3	4	5 - Most preferred
No AI assistance	☐	☐	☐	☐	☐
Chat Mode: A model that can only provide syntax help without access to your code	☐	☐	☐	☐	☐
Agent Mode: A model that can directly access and edit your code	☐	☐	☐	☐	☐
The option to switch between Chat Mode and Agent Mode	☐	☐	☐	☐	☐

10. Was there anything else you liked or disliked about your interaction with the AI during this task? (Optional)

0 chars

Feel free to share anything you liked or disliked.

Continue

Figure 14: Example questions after the initial task submission shown to users in our interface for self-reporting usefulness, understanding, and preferences (§6.1).

Task 1 - My Preview...CodeTime Left: 45:16Submit Project

Project-Specific Questions

Before submitting, please answer the following questions so we can understand your AI usage! Try your best to answer each question. You won't be able to look back at your code.

You may have to scroll down to see all of the questions and the button to proceed.

1. Which of the following features exist in your website? It is possible that all of these or none of these exist.

☐ The board uses CSS Grid with 4px gaps between cells.

☐ The game detects five-in-a-row wins horizontally or vertically.

☐ The page shows a header titled "Zic-Zac-Zoe".

☐ Clicking a cell logs the clicked cell index to the console.

2. Which of these selectors exist in your CSS stylesheet? Each option shows a class or ID selector and a description of what it modifies. It is possible that all of these or none of these exist.

☐ `.board` : This selector styles the main game board container.

☐ `.tile` : This selector styles a single tile element in grid.

☐ `#gameBoard` : This selector targets the primary board wrapper element.

☐ `.cell` : This selector styles individual grid cells within the board.

3. Which of the following JavaScript functions exist in your code? Each option shows a function name and a description of its implementation. It is possible that all of these or none of these exist.

☐ `initializeBoard()` : This function sets up the board for a new game.

☐ `handleCellClick()` : This function processes a cell click and updates state.

☐ `renderBoard()` : This function logs a message displaying the board.

☐ `onMove()` : This function plays a move and renders the board.

Submit Project

Figure 15: Example recall question in the comprehension assessment (§3.3): selecting which functions, elements, and selectors exist in the users' code.

Task 1

My Preview

Code

Time Left: 78:21

Submit Project

Project-Specific Questions

Question 2 of 8

Before submitting, please answer the following questions so we can understand your AI usage! Try your best to answer each question. You won't be able to look back at your code.

You may have to scroll down to see all of the questions and the button to proceed.

1. Exactly one of these two pairs of code blocks (HTML + JavaScript) is from your project. Which one is yours? The ◆ symbol indicates line changes.

Left Block (HTML):

◆ 1

<p id="game-status" class="status"></p>

Right Block (HTML):

◆ 1

<p id="status" class="status"></p>

Left Block (JavaScript):

◆ 1

const statusElement = document.getElementById("game-status");

2 ...

3 statusElement.textContent = `Current turn: \${player}`;

Right Block (JavaScript):

◆ 1

const statusElement = document.getElementById("status");

2 ...

3 statusElement.textContent = `Current turn: \${player}`;

☒ The Left Code is Mine

☐ The Right Code is Mine

2. Exactly one of these two CSS code blocks is from your project. Which one is yours? The ◆ symbol indicates line changes.

Left block:

1

body {

2 margin: 10px;

3 min-height: 100vh;

4 min-width: 100vw;

5 display: flex;

6 align-items: center;

◆ 7 justify-content: center;

8 }

Right block:

1

body {

2 margin: 10px;

3 min-height: 100vh;

4 min-width: 100vw;

5 display: flex;

6 align-items: center;

◆ 7 margin-left: auto;

◆ 8 margin-right: auto;

9 }

☒ The Left Code is Mine

☐ The Right Code is Mine

3. Exactly one of these two JS code blocks is from your project. Which one is yours? The ◆ symbol indicates line changes.

Left block:

1

function renderBoard() {

2 console.log("Displaying the board");

3 // Update the UI to reflect the board state

◆ 4 for (let i = 0; i < 25; i++) {

◆ 5 const row = Math.floor(i / 5);

◆ 6 const col = i % 5;

◆ 7 cells[i].textContent = board[row][col];

8 }

9 }

10 // Show whose turn it is or winner

11 const winner = checkWinner();

12 if (statusElement) {

13 if (winner) {

14 statusElement.textContent = `Player \${winner} won!`;

15 } else {

16 let player;

17 if (Math.floor(moveNumber / 2) % 2 === 0) {

18 player = "A";

19 } else {

20 player = "B";

21 }

Right block:

1

function renderBoard() {

2 console.log("Displaying the board");

3 // Update the UI to reflect the board state

◆ 4 for (let row = 0; row < 5; row++) {

◆ 5 for (let col = 0; col < 5; col++) {

◆ 6 const i = row * 5 + col;

◆ 7 cells[i].textContent = board[row][col];

◆ 8 }

9 }

10 }

11 // Show whose turn it is or winner

12 const winner = checkWinner();

13 if (statusElement) {

14 if (winner) {

15 statusElement.textContent = `Player \${winner} won!`;

16 } else {

17 let player;

18 if (Math.floor(moveNumber / 2) % 2 === 0) {

19 player = "A";

20 } else {

21 player = "B";

Figure 16: Example recall question in the comprehension assessment (§3.3): selecting which of two logically-equivalent snippets the user created.

Task 30.1

My Preview 30.1

Submit

Project-Specific Questions

Question 3 of 8

Before submitting, please answer the following questions so we can understand your AI usage! Try your best to answer each question. You won't be able to look back at your code.

You may have to scroll down to see all of the questions and the button to proceed.

1. Which of these options best describes how the JavaScript code below uses the HTML element `<p id="status" class="status"></p>`?

HTML:

```
1 <p id="status" class="status"></p>
```

JavaScript:

```
1 const statusElement = document.getElementById("status");
2 ...
3 statusElement.textContent = `Current turn: ${player}`;
```

- ☐ The JavaScript updates this element to display the game status
- ☐ The JavaScript uses this element to store the game board state for future access
- ☐ The JavaScript updates this element to determine which player symbol appears next
- ☐ The JavaScript listens for changes to this element to detect player moves

Continue

Figure 17: Example code reasoning in the comprehension assessment (§3.3): identifying the purpose of a function in the users' code.







Task 3.1My Preview 3.1

←

→

Submit

Project-Specific Questions

Question 4 of 8

Before submitting, please answer the following questions so we can understand your AI usage! Try your best to answer each question. You won't be able to look back at your code.

You may have to scroll down to see all of the questions and the button to proceed.

1. In the HTML snippet below, the JavaScript selects the status element using the `id` identifier. If the `id` on the HTML element was changed from `status` to `game-status` but the rest of the website remained the same, what would most likely happen?

HTML:

```
1 <p id="status" class="status"></p>
```

JavaScript:

```
1 const statusElement = document.getElementById("status");
2 ...
3 statusElement.textContent = `Current turn: ${player}`;
```

☐ The player turn and win detection logic would change

☐ The status element would disappear from the page

☐ The status element would be able to update

☐ The status element would never update

Continue

Figure 18: Example code reasoning in the comprehension assessment (§3.3): identifying how the user's game would change if their code changed.

Rubric A.1: Initial Task

- The first symbol placed in a new game is A (the game starts with Player A's turn). If there is no turn placing logic, this rule is automatically violated.
- Turns alternate between Player A and Player B. If there is no turn placing logic, this rule is automatically violated.
- Each player places exactly two symbols per turn before the turn switches. If there is no turn placing logic, this rule is automatically violated.
- Clicking an empty square places the correct symbol (A or B) based on whose turn it is. If there is no turn placing logic, this rule is automatically violated.
- Players cannot place a symbol on a square that already contains a symbol. If there is no turn placing logic, this rule is automatically violated.
- After a game ends, no additional symbols can be placed. If there is no win detection logic or no end game states can be reached, this rule is automatically violated"
- The game correctly detects horizontal five-in-a-row win conditions. If there is no win detection logic, this rule is automatically violated.
- The game correctly detects vertical five-in-a-row win conditions. If there is no win detection logic, this rule is automatically violated.
- The game has no other win conditions (e.g., diagonal wins). If there is no win detection logic, this rule is automatically violated.", "If the board fills with no winner, the game ends in a tie. If there is no win detection logic, this rule is automatically violated.
- A status element visually appears on the page prefixed by "Status:". Minor variations of 'Status' are fine and should NOT be considered violations.
- The status element correctly displays whose turn it is. If there is no turn changing logic, this rule is automatically violated.
- The status element correctly displays the game outcome (Player A wins, Player B wins, or Tie Game). If there is no game detection logic, this rule is automatically violated.
- The entire web page is centered horizontally. If it is also centered vertically, it is NOT considered a violation.
- Horizontal centering of the page is implemented by modifying the CSS file. If the page is not centered horizontally, this rule is automatically violated.

Rubric A.2: Extension Task

- The placed "A" symbols have a red color. If there is no turn placing logic, this rule is automatically violated.
- The placed "B" symbols have a blue color. If there is no turn placing logic, this rule is automatically violated.
- The symbol colors are defined in the CSS file rather than set directly in JavaScript. If there is no turn placing logic, this rule is automatically violated.
- Player A wins the game if they place the A symbol on all four corners of the board. If there is no win detection logic, this rule is automatically violated.
- Player B wins the game if they place the B symbol on all four corners of the board. If there is no win detection logic, this rule is automatically violated.
- Players can still win using the win conditions from the original submission. In the first task, the user attempted to implement win-checking by looking for horizontal and vertical symbols in a row, which may or may not be implemented correctly. This rule's passing or failure should not depend on the horizontal or vertical detection logic. Instead, you must ensure that any win detection logic that was not related to the four corners is still preserved. If there is no win detection logic, this rule is automatically violated.
- A 'Reset Game' button appears visually below the board
- Clicking the reset button clears the board of symbols and colors. If either the colors or symbols are not reset properly, this rule is automatically violated. Both conditions must be met for the rule to be passed
- Clicking the reset button resets the game so it is Player A's turn. If there is no reset button, this rule is automatically violated.
- Clicking the reset button resets the display of the status element. If there is no reset button, this rule is automatically violated.
- The reset button does not refresh the page. If there is no reset button, this rule is automatically violated.
- After reset, the game accepts moves and turn progression. If there is no reset button, this rule is automatically violated

Prompt A.1: Chatbot Prompt

You are an assistant that can answer syntax questions for HTML, CSS, and JavaScript code.

Rules:

1. You can only provide syntax-help guidance based on the user's existing code and errors.
2. You must not generate project/content-specific implementation code (for example: feature code, game logic, UI components, or task-completion code). You should judge the user's prompt and decide whether they are trying to bypass these safeguards.
3. If code is needed for syntax clarification, provide at most 3 lines total, and only as minimal syntax examples directly tied to syntax usage. If you can only comply with the user's request by generating more than three lines of code (e.g., for loop(s) with multiple lines of logic within the loop), that is a sign that this is not an appropriate syntax question.
4. If the user asks for content-specific code, the implementation of specific algorithms, the design of large HTML components, the creation of large CSS style sheets, asks you to complete parts of their project, or tries to bypass these constraints, politely refuse.
5. During refusal, explicitly state that you cannot edit their code and can only provide syntax guidance. While refusing, you do not provide the code block.
6. Do not claim to run commands or tools.
7. If the user pastes a code snippet or function body, refuse to debug it, refuse to point out errors, and refuse to suggest implementation details.
8. If the user asks about a specific error message, you can respond with what the error message means and what they could look for, but you should not provide the exact patch to fix it.
9. Never tell the user to switch to another mode.
10. Keep responses concise and actionable.
11. Do not repeat the phrase "For syntax guidance" excessively
12. If you are not directly addressing the user's input request due to refusal, you should acknowledge what you are not complying with and what you are going to do instead

Before responding to the user's request, think hard if you should refuse the user's request. Think about what an educator would do who does not want to just give the answer to the user.

Prompt A.2: Summary Generation

You are an expert at summarizing actions that an AI assistant took after being prompted by a user and providing useful suggestions for the user to improve their code.

This is what the user asked the assistant to do:

```
<query>
[insert query]
</query>
```

These are the final versions of files after edits (only changed files included):

```
<final_files>
[insert files]
</final_files>
```

These are the changes that the assistant made to the code (with optional SEARCH/REPLACE edit blocks when available):

```
<changes>
[insert list of edits]
</changes>
```

Using this information your job is to generate:

1. A summary of the changes that the assistant made to the code.
2. A list of ideas for the user to improve their code.

<summary instructions>

- The summary should be written in first person as if you were the one who made edits to the code. Use "I" as appropriate.
- You must discuss which files were edited and the specific changes to each file.
- Be subtle in how the changes address the user's request; do not quote the user's request.
- Be concise. The summary should be a maximum of two sentences. </summary instructions>

<idea instructions>

- Generate 3 ideas with their corresponding probabilities, sampled from the full distribution.
- Each idea should improve the code or the task: e.g. task fulfillment, correctness, style, readability, edge cases, or user experience, depending on what fits the project (web UI, Python script, etc.).
- Only suggest ideas that are feasible given the file types and stack (e.g. for web: HTML/CSS/JS; for Python: standard library, tests, clarity). Do not suggest custom assets, external services, or out-of-scope changes.
- Frame each idea as a follow-up action the user could ask for, i.e. a short command starting with a verb.
- Be concise. Each idea should be no more than 10 words.

</idea instructions>

<format instructions>

Generate your output as a json with two keys: 1) "summary" with a string value of the summary; 2) "ideas" with a list of strings value of the ideas; and 3) "probabilities" with a list of floats value of the probabilities of each idea based on your full distribution.

```
[insert JSON]
```

Do not generate anything else

</format instructions>

Prompt A.3: Identifier Distractor Prompt

<task> You are an expert at generating function names that do not exist in a user's code but plausibly could.

Given the existing function names and their implementations below, generate exactly four function names that mimic the style and domain of the existing code but do not actually exist. Use the implementations to understand what the code does (e.g. game turns, setup, UI) so your fake names are plausible. For example, if the code has "playTurn" and "setupLogic", you could generate "endTurn", "setupGame", "resetBoard". We will show these to users and ask them to identify which names exist and which do not, testing their comprehension of their own code. </task>

Here are the existing functions (name and implementation):

<functions> [insert functions] </functions>

<function requirements>

- Mimic the style and domain of the existing function names; use the implementations to inform plausible fake names.
- None of the generated function names should be the same as the existing function names: [insert function names]. This is extremely important.
- You may generate: 1) names for features that do not exist in the code; 2) wrapper/helper names that do not exist; 3) names that suggest further decomposition of the real logic.

</function requirements>

<format>

Generate your output as a JSON with the key "fake_function_names" and the value being an array of exactly four function names as strings:

[insert JSON]

Do not generate anything else. </format>

Prompt A.4: Snippet Distractor Prompt

<task>

Generate a distractor for this code block.

- Introduce one noticeable but logically equivalent difference such that someone who did not actually implement this code themselves may not realize the difference
- Do not add obvious stylistic differences or artifacts. For example, if the original code block did not have comments, you should not add comments to the distractor block. If the original code block did have comments, you should preserve them.

</task>

<code> code </code>

<html-priority-targets>

For HTML distractors, prioritize the following changes in this order:

- switch the label of the element's ID to something that this user could plausibly specify. this distractor should match the style of the user's
- switch the label of the element's class name to something that this user could plausibly specify. this distractor should match the style of the user's
- any other change that is noticeable but different

</html-priority-targets>

<task-specific-css-focus>

For zic_zac_zoe centering-related CSS, generate a different but logically equivalent way that the user could have centered their code horizontally, i.e., by altering different attributes than what the user's code did. Try to use one that is similar to what they did in style. For example, if they used justify-content: center, you could use align-items: center and switching the flexbox direction (and vice versa).

There are different ways to center the component, but the distractor should focus on one that appears simple. If you add too many extra lines compared to the original its a clear giveaway.

If this is not possible, introduce some other logically equivalent difference (but they should not be identical)

</task-specific-css-focus>

<task-specific-js-focus>

For zic_zac_zoe board rendering implementations, add an logically equivalent change related to indexing. Prefer one of the distractors in this order:

- If the user loops over a 2D matrix (e.g., the 2D 'board' based on rows and columns) and then converts it into a 1D matrix index (access 'cells' via 5*i+j), swap the approach: loop over the 1D list (cells) and map it into a 2D matrix indices (board). The vice versa also applies
- Make sure the style is EXACTLY the same between the two versions, including comments, indentation, and newlines. Try to use a more conventional style for both.
- If this 2D vs 1D change is not possible because the user has not implemented the function this way, introduce some other logically equivalent change of the function (but they should not be identical)

</task-specific-js-focus>

<format> Return strict JSON only: [insert JSON] </format>

Prompt A.5: Question Personalization Prompt

<task> You are validating whether a multiple-choice question can be asked about a user's code snippet. You MUST do two things:

- 1) Rewrite the question so it aligns with the provided question template and code details.
- 2) Validate the answer options, and adapt their wording only if needed for this participant's code.

Interpret the code literally. Do not infer intended behavior beyond what the code actually does or attempts to do. The generated question is meant to test the understanding of the user's own code.

Question-rewrite rules:

- Try to adhere to the style of the question_template.
- Fill in concrete details from the participant's code (e.g. element IDs, function names, selectors) where the template has placeholders.
- Write a single clear question that stands on its own; the snippet will be shown right after it
- Make sure the question is extremely clear and easy to understand

Validation rule:

- The designated gold option must be correct for the user's actual code.
- Every distractor option must be incorrect for the user's actual code.
- The gold option must describe what the code ACTUALLY does, not the intended behavior

****Abstention policy.**** A question should be selected when the participant's code contains a recognizable artifact related to the target feature. If no such artifact exists, return status = "abstain". Use the following guidelines:

- HTML questions: Select the question if the code includes a status element intended to display game status on the page (e.g., a <p>, <div>, or similar element that appears to hold status text) and JavaScript code that attempts to access that element (e.g., selectElementById, querySelector, ...)
- CSS questions: Select the question if the code includes a CSS rule that appears to attempt positioning or aligning the board or page layout (i.e., centering rules like align-items or justify-content). The implementation may be incomplete or incorrect. If the user did not change their code based on the starter implementation, then you MUST abstain.
- JavaScript questions: Select the question if the code includes a function or logic that renders or populate the game board on the page (e.g., iterating over the board state and updating DOM elements). The implementation may be incomplete or incorrect. If this function (initially called renderBoard()) does not exist or was not attempted (blank or has a print statement), abstain.

If none of these artifacts are present for the relevant question type, return status = "abstain".

Option-adaptation rules:

- Start from the provided sample gold answer and sample distractors.
- Keep the original wording when it already fits the participant's code.
- If wording does not fit, adapt it so one option is clearly correct and three are clearly incorrect for this participant's implementation.
- Avoid introducing stylistic cues: keep all four options similar in length (no more than five word differences), tone, specificity, and structure. In particular, all distractors must be very similar in length to each other (and the gold option should be similar too) so that length does not cue the answer. Do not add extra punctuation like parentheses or semi-colons on individual answers.
- None of the options should include words that relate to "change"; this is especially important in the counterfactual-style questions. We don't want to cue the user based on how their implementation current works. So instead of saying "Your board would stay centered" or this would "no longer" work, you could say "Your board would be centered"
- Remember: one option must stay correct and three must stay incorrect.

</task>

<question_spec> [insert question template] </question_spec>

<full_website_context> [insert website] </full_website_context>

<selected_snippet> [insert snippet] </selected_snippet>

The question template below provides an example question choices for a WORKING implementation. If the user's implementation is correct, you can adapt this question directly. If their implementation is incomplete or incorrect, you will have to adapt the options to make sure the gold answer is correct and the distractors are objectively wrong.

<provided_option_templates>

- sample_gold_answer: [insert gold answer]

- sample_distractors:

- [insert distractor 1]

</provided_option_templates>

Remember, if you are changing the gold answer, keep the word length differences minimal! (less than 5 words)

<format> Return strict JSON only. Also provide a short reason explaining why the question was selected or why the feature was absent in a "reason" key, and how you made any changes: [insert JSON] </format>

Prompt A.6: Judge Prompt

You are an expert at evaluating whether user's code adheres to rules from a rubric used as grading criteria.

A user was asked to create a fictional game called "Zic-Zac-Zoe" using HTML, CSS, and JavaScript. They had to implement these specifications:

<specifications the user saw> [insert requirements] </specifications the user saw>

They finished the assignment and submitted the following code:

<code> [insert code] </code>

Now, as educators, we must evaluate their code to give them a final grade. We constructed a rubric from the zic-zac-zoe task, and now we will score each rule one at a time. Here is the current rule from the full rubric that you are assessing:

<rule> [insert single rubric rule] </rule>

Your task is to judge whether the user's submitted code ****passes or violates this rule****. Here are some guidelines for evaluation:

<evaluation instructions>

- Grade the code like an educator. Do not allow the user to be clever and bypass the rules; they must make a "good-faith" effort to follow the rule as intended
- There is no partial credit. Either the rule is "passed" or "violated"
- Grade the website as if you were looking at it visually. For this reason, if the code has syntax errors that would stop the whole site or certain parts from rendering, then you should not consider the unrendered parts as properly implemented. For example, if the JavaScript code has a syntax error, then none of the logic would run and should not be evaluated. We know that the HTML, CSS, and JavaScript are correctly linked together so cross-file integration will not be an issue.

</evaluation instructions>

<format>

Return your output as valid JSON in the following format:

[insert JSON]

Do not include anything else.

</format>

Prompt A.7: Cluster Labeling Prompt

You are an expert at classifying text into its most relevant cluster

You will analyze user prompting strategies as they work with an AI assistant to develop a game of tic-tac-toe. After the user submits their prompt, the agent directly modifies the user's code to adhere to that request. Your job is to classify from a list of cluster names, which best applies to the input prompt.

These were the exact rubric items the user had to meet with their submission:

<rubric> [insert rubric] </rubric>

Here are your instructions:

<instructions>

- Choose the single best cluster for the action from the list of clusters. Do not invent new cluster titles.
- Prefer a specific existing cluster whenever possible.
- Use "Other" only when no listed cluster is a reasonable semantic fit.
- Ensure you reference the <rubric> when making your decision
- Keep explanation concise but concrete and evidence-based.

</instructions>

Here are the name of clusters

<cluster names>

1. copy: the user directly copies one or more of the rubric requirements verbatim or nearly verbatim
2. rephrased: the user rephrases one of the rubric requirements in their own words, but still using natural language. Prompts in other languages fall into this category
3. technical: the user mentions specific HTML, CSS, or JavaScript syntax (e.g., HTML tags, function names, or CSS selectors/identifiers), rather than using pure natural language text
4. exploratory: the user asks the AI to summarize the codebase or about its abilities
5. debugging: the users asks for the AI to refine or debug an initial mistake

</cluster names>

Here is the input prompt you must classify:

<prompt> [insert prompt] </prompt>

<format>

Return a JSON with two keys: 1) a string "cluster_title" which is the name of the cluster you chose to classify; and 2) a string "explanation" which explains your decision

[insert JSON]

Return ONLY the JSON object without extra text

Prompt A.8: Cluster Chatbot Labeling Prompt

You are an expert at classifying text into its most relevant cluster

You will analyze user prompting strategies as they work with an AI assistant to develop a game of tic-tac-toe. After the user submits their prompt, the chatbot responds to that request, but it only helps the user by returning high-level syntax guidance; it never executes tasks on the user's behalf. Your job is to classify from a list of cluster names, which best applies to the input prompt.

These were the exact rubric items the user had to meet with their submission:

<rubric> [insert rubric] </rubric>

Here are your instructions:

<instructions>

- Choose the single best cluster for the action from the list of clusters. Do not invent new cluster titles.
- Prefer a specific existing cluster whenever possible.
- Use "Other" only when no listed cluster is a reasonable semantic fit.
- Ensure you reference the <rubric> when making your decision
- Keep explanation concise but concrete and evidence-based.

</instructions>

Here are the name of clusters

<cluster names>

1. syntax_help: the user asks about basic HTML, CSS, or JavaScript syntax, operators, methods, DOM APIs, selectors, attributes, or language mechanics
2. design_help: the user asks how to structure the task, reason through the game rules, manage turns, check win/tie conditions, or decompose the problem without requesting a full solution
3. snippet: the user asks for a specific localized code fragment, function, condition, loop body, counter, or partial implementation step
4. debugging: the user reports an error, broken behavior, unexpected output, layout issue, or asks the AI to fix/refine something that is not working
5. clarification: the user asks whether a prior suggestion is correct, where code should go, what a term means in context, or how to adapt a previous answer
6. jailbreaking: the user asks for a complete implementation, tries to bypass the chatbot's limits, copies or reframes the whole assignment as a request for finished code, or uses pressure tactics to obtain prohibited help
7. urgency: the user expresses frustration, distress, profanity, dissatisfaction, time pressure, or emotional escalation; this can be used as a secondary label alongside one of the other categories

</cluster names>

Here is the input prompt you must classify:

<prompt> [insert prompt] </prompt>

<format>

Return a JSON with two keys: 1) a string "cluster_title" which is the name of the cluster you chose to classify; and 2) a string "explanation" which explains your decision

[insert JSON]

Return ONLY the JSON object without extra text

</format>