Investigating Think-Aloud Speech as Context for AI Code Generation

Chetan Goenka
cgoenka@berkeley.edu
UC Berkeley
Berkeley, California, USA

J.D. Zamfirescu-Pereira
zamfi@cs.ucla.edu
UCLA
Los Angeles, California, USA

Bjoern Hartmann
bjoern@eecs.berkeley.edu
UC Berkeley
Berkeley, California, USA

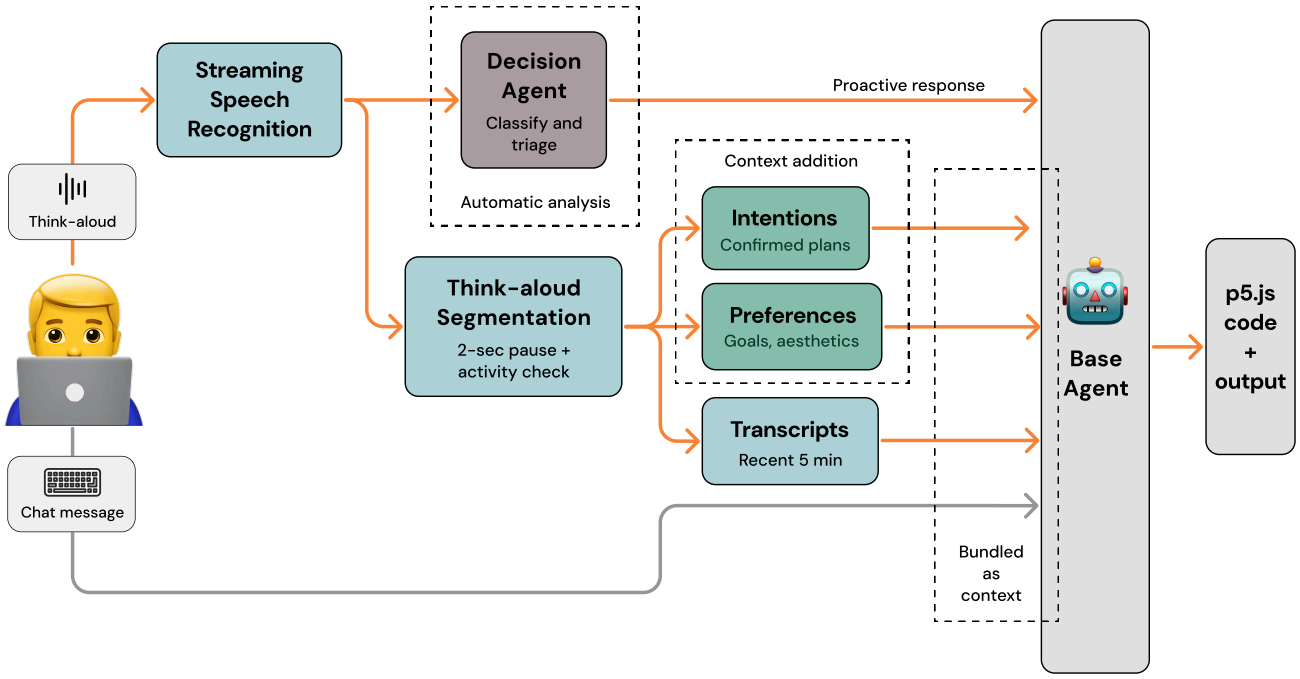

Figure 1: Overview of the TAP system pipeline. The developer's speech is continuously transcribed and segmented into think-aloud episodes. These episodes are processed through two complementary pathways: *autonomous analysis* (upper-top), where a decision agent evaluates whether a proactive response is warranted, and *context addition* (lower-top), where recent transcripts, extracted preferences, and confirmed intentions are bundled alongside the developer's typed chat messages before reaching the base conversation agent.

Abstract

AI-powered programming tools generate code from explicit text inputs: prompts and surrounding code context. Yet these inputs capture only what developers choose to externalize in writing, missing the continuous stream of design reasoning, evolving preferences, and spontaneous judgments that shape programming decisions but rarely make it into goal-directed, written instructions. We present an IDE extension that uses think-aloud speech as a new channel of context for AI code generation. The system continuously captures developers' think-aloud, maintaining a running context of their design reasoning, preferences, and intentions that shapes every AI action alongside existing chat-based prompts and code. Through a within-subjects study with 11 developers, we find that think-aloud context changed what the AI assistant produced: it incorporated preferences and design goals that participants never typed, surfaced verbal frustrations as actionable suggestions, and—in rare cases—proactively fixed errors detected from speech alone. These effects were most visible when participants left design space open in their typed requests, suggesting that think-aloud context complements rather than replaces typed communication. Participants

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

in the think-aloud condition trended toward a stronger sense of collaboration with the AI assistant, and the majority preferred it for creative prototyping—though opinions on the naturalness of thinking aloud while coding were mixed. Our findings suggest that think-aloud speech can provide AI tools with rich preference and intent information that would otherwise go untyped.

Keywords

AI-assisted programming; Think-Aloud Programming; Voice Interactions

ACM Reference Format:
Chetan Goenka, J.D. Zamfirescu-Pereira, and Bjoern Hartmann. 2018. Investigating Think-Aloud Speech as Context for AI Code Generation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

An explosion of AI-powered code generation tools has begun to reshape how programmers at all experience levels author code, with some estimates pointing to 50%+ of code at major technology companies being authored by AI. These tools overwhelmingly rely only on direct text-based prompting to capture programmer intent, limiting their scope of communicated intent to the explicit instructions programmers enumerate. When collaborating with humans, however, human programmers also have access to a less-formal, continuous stream of design reasoning, evolving preferences, and spontaneous judgments that are often spoken aloud, as in pair programming. Though rarely articulated to AI code generators, the data in this stream could offer additional context for AI tools to make better decisions that are better aligned with the evolving intent of the human programmer.

In this work, we present one way to implement explicit capture of this stream: TAP, a think-aloud IDE that captures (via audio recording and transcription) a stream of think-aloud speech containing reasoning, reflection, and preference information. This transcription is then included in the context window as alongside requests for code generation, as well as analyzed for any urgent requests which are then responded to immediately (rather than waiting for a new text chat request to be sent). Drawing on Krosnick et al.'s paradigm of *Think Aloud Computing* [8], we expect these transcript slices to offer AI models additional useful intent context around the explicit requests users make—just as human collaborators working on code together will ground, examine, and justify their evolving intent in real-time with each other.

Through a $N = 11$ within-subjects lab study of developers, we explore how TAP's think-aloud captures influence how a chat-based coding session unfolds compared to an identical IDE that ignored think aloud content. We find that think aloud content is, in fact, used by the AI both implicitly and explicitly, and that this content is helpful for generating preference and intention models of preferences and intentions that aren't explicitly written down. These uses are overall received positively by our participants, who feel a stronger sense of collaboration with the AI assistant when using think aloud. These effects appear greater when participants leave the design space open while working through a task. Our findings suggest that think aloud content can thus provide a rich additional channel for preference and intent information that might otherwise have gone "untyped."

Our work makes several contributions:

(1) TAP, an IDE that includes think aloud capture and use of that data to support goal alignment between TAP's human user and its AI programmers.
(2) An preliminary investigation of whether, and when, think aloud data can be useful, including an exploration of a fwe different ways of using this data: to proactively respond, to update a preference and intention model, or to provide additional nuance to or a spontaneous evaluation of the system under design.

2 Related Work

TAP's design draws on several threads of recent research, in particular on AI-powered code generation, pair programming, and think-aloud computing and protocols more generally.

2.1 AI-Powered Code Generation

From Vibe Coding [1, 14] to purpose-built AI-powered IDEs both real (e.g., [7, 10, 18]) and speculative (e.g., [15]), to studies of expert and non-expert programmer behavior (e.g., [2, 6, 11, 13, 16]), much has been written about the methods and experiences of programming with AI assistance. These threads converge on some common themes: LLMs enable intent specification at many different levels of abstraction, but this comes with challenges; understanding what generated code does is not always trivial (nor always desired or required); design requirements change over time, and systems must adapt to these.

Through this work, we aim to complement these themes by investigating how a common human-human interaction paradigm, think-aloud practice, can aid in intent elicitation and capture.

2.2 Pair Programming, Think Aloud, and Collaboration

In human-human contexts, continuous verbalization, as in the driver-navigator dynamics of pair programming [3, 17] and think-aloud protocol analysis [5, 9], serves as a primary mechanism for making problem-solving processes observable. These vocalized streams surface hypotheses, structural planning, and intermediate debugging steps that are often ignored by discrete code commits or in formal documentation [4]. Recent advancements in conversational AI and chat-integrated IDEs [12] have begun to incorporate natural language dynamics into the programming environment, enabling developers to iteratively discuss and refine code through synchronous, turn-based dialogue. Studies evaluating these interactions note that the need to formulate explicit, well-structured prompts and shift attention between writing code and managing dialogue can introduce measurable cognitive load and context switching [2, 16]. Our work explores the intersection of these domains, investigating how supplementing discrete conversational turns with continuous, ambient verbalization, mirroring traditional think-aloud methods, might provide AI assistants with a real-time stream of programmer intent while mitigating the cognitive overhead of active prompt formulation.

3 System Design

TAP is a think-aloud system built within an existing AI-powered web-based IDE [anonymous for review] for programming `p5.js` sketches. This IDE provides a chat-based coding agent that can directly read and edit developers' code, a code editor with in-line code diffs, and a live output panel and console logs. TAP extends this IDE with a pipeline that continuously listens to developers' think-aloud speech, processes it to extract actionable context, and incorporates that context into the requests made of the AI coding assistant. The system's design was iteratively refined through repeated testing with two CS PhD students over several weeks, which informed key decisions throughout the TAP system design described below.

TAP utilizes think-aloud speech through two complementary pathways:

(1) Automatic Analysis: when there is a pause in the developer's think-aloud, the system evaluates whether a proactive response from the AI assistant is warranted. If so, this response is provided to the developer.
(2) Think-aloud Context Addition: collected think-aloud transcripts and extracted developer preferences plus intentions are always supplied with with typed chat messages, giving the AI assistant access to the developer's recent thinking.

Both pathways share a five-stage pipeline: real-time speech transcription, pause-based processing, classification, preference and intention extraction, and context incorporation into the main conversation.

3.1 Speech Capture and Processing

TAP maintains a continuous streaming connection to a cloud-based speech recognition service[1], transcribing the developer's think-aloud in real-time. A finalized transcript accumulates as the developer continues speaking. We display a microphone icon with added visual indication to provide ambient awareness that the system is listening.

The system segments think-aloud into coherent episodes using pause-based processing: a 2-second pause in the developer's speaking triggers automatic analysis of the collected think-aloud transcript. Through iterative testing we found this duration balanced responsiveness with avoiding premature segmentation of developers who briefly pause mid-thought. Before initiating the automatic analysis, the system checks a multi-channel activity monitor that tracks the recency of speech, typing, and code editing. If the developer is actively typing or has text in the chat input field, automatic analysis is suppressed, with the assumption being that a typed message, a deliberate, goal-driven communicative act, will take precedence over the speech content. This suppression only applies to the automatic processing pathway. The accumulated think-aloud transcript is still incorporated as context when a typed message is sent. If new speech arrives before the current automatic processing is completed, it is appended to the pending think-aloud segment and the in-progress analysis is canceled, ensuring the system always evaluates complete recent context.

[1]We used Deepgram's Nova-3 model

3.2 Classification and Proactive Response

A dedicated decision agent classifies each processed think-aloud segment to determine whether the AI assistant should respond proactively. The agent receives the collected think-aloud transcript along with recent conversational context and a reaction annotation: a flag indicating whether the think-aloud arrived within 30 seconds of an AI assistant code change, which helps distinguish evaluative commentary ("hmm, that doesn't seem right") from independent thought. It returns a structured decision: whether to respond, at what urgency (`immediate`, `can_wait`, or `none`), and a category (`question`, `error`, `stuck`, `confusion`, or `observation`).

The agent is prompted to adopt conservative behavior, defaulting to silence. TAP uses speech as an ambient context channel, not a voice command interface: the developer is thinking out loud, not issuing instructions. Most think-aloud speech (self-directed questions, planning statements, self-encouragement) does not warrant a response and risks interruptions, disturbing the developer's flow. Early iterations confirmed this concern: without strong constraints, the system triggered frequent false positives that our pilot test developers found disruptive. The decision agent prompt therefore explicitly provides common think-aloud patterns to ignore, reserving proactive responses for clear knowledge gaps, error reports, and genuine stuck states. In our final version of the system, proactive responses were rare, most often triggered when developers mentioned bugs or errors. Post-decision filtering applies category-specific confidence thresholds, for example, questions require higher confidence than error reports, since self-directed questions are common during think-aloud. If a segment passes all filters, the collected transcripts are formatted as a timestamped message and routed to the base conversation agent, which is instructed that this message originates from the developer's think-aloud rather than a direct chat request.

3.3 Preference and Intention Modeling

Two extraction agents run in parallel alongside classification, each building a running model from the think-aloud stream that persists across the session.

Preference extraction captures the developer's aesthetic choices, creative goals, feature ideas, and satisfaction or dissatisfaction signals. It runs every three think-aloud segments, or immediately when a segment is annotated as a reaction to a recent code change since reactions are especially rich in preference signal (e.g., "I like how those colors look" or "the UI feels too rigid"). Each extraction returns the full updated preference state with incremental semantics: new preferences are added, contradicted ones removed, and stable entries preserved.

Intention extraction captures confirmed action plans — things the developer has decided to do but has not yet asked the AI assistant for help with (e.g., "I'm going to add a larger set of words to this flashcards application"). During iterative design testing, we observed our pilot developers frequently verbalizing such plans without ever typing them, representing actionable signal that was otherwise lost. The extraction agent uses strict inclusion criteria, distinguishing committed plans ("next I'll work on the background") from vague exploration ("maybe I could..."), observations, and questions. Extracted intentions follow a life-cycle: they begin as `pending`,

are marked `delivered` once incorporated into a conversation, and expire after 20 minutes to prevent stale plans from persisting.

3.4 Think-aloud Context Incorporation

When the developer sends a typed chat message, TAP constructs a context bundle that is supplied alongside the message to the AI coding assistant. This bundle contains three components: 1) recent think-aloud transcripts from the last five minutes, split into new and already-addressed thoughts, 2) a summary of the current preferences, and 3) saved pending intentions, filtered against the typed message to avoid repeating what the developer is already asking for. Each component is replaced on every message, so only the most recent snapshot is retained in conversation history.

The base conversation agent's prompt instructs it to apply preferences proactively, incorporating them into generated code and suggestions, weaving in confirmed intentions naturally (e.g., "Building on your idea to..."); and to avoid meta-commentary about the think-aloud process. The goal is for the AI to behave as though it has been present in the room, aware of the developer's evolving thinking without drawing attention to the mechanism.

4 User Study

We conducted a within-subjects study to answer two research questions:

RQ1: Does capturing developers' live think-aloud speech as context for AI code generation improve intent capture compared to chat-only interaction?

RQ2: Do developers prefer the outcomes and interaction model of think-aloud-assisted coding over chat-only?

The underlying AI assistant[2] was identical across conditions. Following the AI-powered IDE's design [anonymous for review], the conversation agent guided participants through a user-centered design process, identifying target users, evaluating needs, and proposing approaches before generating code. The agent did not provide in-line code suggestions; instead, it engaged conversationally and then directly wrote or modified the project code. This interaction model makes the communication between developer and AI central to the quality of the code, and thus a meaningful way for studying the impact of think-aloud speech context.

4.1 Participants

We recruited 11 participants (6 female, 4 male, and 1 undisclosed) from university mailing lists and lab networks, aged 18–34. Programming experience was required and it was distributed as 1–2 years (3), 3-5 years (5), and 6–10 years (3). Almost all were regular users of AI coding tools: five used them daily and four several times per week. Seven had some JavaScript familiarity and only one had used p5.js before. None had previously used our system. Participants were compensated $30 for a 75-minute remote session conducted over Zoom.

4.2 Tasks and Conditions

Participants completed two coding tasks under two conditions. In the **think-aloud (TA) condition**, TAP's think-aloud capture pipeline was active; the system continuously transcribed participants' think-aloud, extracted preferences and intentions, and incorporated this context into every AI interaction. In the **control (C) condition**, think-aloud capture was disabled and participants interacted with the AI exclusively through typed chat.

Participants were asked to think aloud in both conditions. Before the TA condition task, the facilitator additionally informed participants that the AI would use their think-aloud as context in its responses. This design was chosen to isolate the effect of the system's use of think-aloud context, rather than the act of thinking aloud itself.

Participants completed two 22-minute p5.js coding tasks, one per condition, designed to be comparable in scope, complexity, and design ambiguity:

Vocabulary Building App - Build an app to help a user learn vocabulary words.

Habit Tracker App - Build an app to help a user track daily habits and see progress over time.

There were no build requirements and the design was intentionally left open-ended so that the user's creativity wasn't constrained, with the assumption that it would elicit more and a wider range of think-aloud directions. The 22-minute task duration, was also a deliberate choice informed by pilot testing. Shorter sessions (15 minutes) cut participants off before they could move past the initial specification stage into iterative refinement. The iteration phase think-aloud had most distinctive signal: reactions, evaluations, and reasoning that rarely appeared in typed messages. The longer duration allowed participants to experience the full arc of building an application from scratch, preserving creative freedom without constraining them to pre-loaded starter code.

We used 2×2 counterbalancing across task–condition pairings. Due to a group assignment error, groups were unevenly distributed (4-4-1-2 across the four groups). Condition order remained nearly balanced (5 TA-first, 6 C-first).

4.3 Procedure

Each 75-minute session began with consent, brief background questions, and a 7-minute tutorial plus sandbox introducing the IDE and its chat-based workflow. Participants then completed the two 22-minute tasks with a post-task questionnaire after each, followed by an open-ended questionnaire-based exit interview comparing the two conditions. The think-aloud context feature was introduced only immediately before the TA condition task and not during the tutorial to avoid priming participants during the control condition. The facilitator described the study as an evaluation of "an AI-powered coding tool" without revealing its focus on think-aloud speech. Participants were told that completing a polished app was not the goal, and if a participant finished early and was unsure how to continue iterating, they were suggested to ask the AI assistant for its ideas on further development.

4.4 Analysis & Evaluation

After each task, participants completed a 10-item questionnaire (5-point Likert scale) spanning five categories: AI alignment and understanding (3 items; e.g., "The AI seemed to understand my goals and intent"), communication and expressiveness (2 items), cognitive

[2]We used OpenAI's GPT-5.4 model for all LLM calls.

load (1 item), collaboration quality (3 items), and outcome satisfaction (1 item). After the TA condition, three additional items assessed the think-aloud experience (naturalness, self-consciousness, and whether thinking out loud helped reasoning). Exit interview questions asked participants to directly compare conditions, including which they preferred and whether the AI understood them better in one condition, followed by open-ended questions on their experience using the system. We also collected screen recordings, AI chat logs, think-aloud transcriptions, code version histories, and additional logs documenting what the AI use as context each turn.

5 Findings

We report findings from our within-subjects study (n=11) comparing the think-aloud (TA) and control (C) conditions. We first present quantitative questionnaire results, then qualitative findings from thematic analysis of TA session logs.

5.1 Quantitative Analysis

Participants rated their task-system experience on 10 Likert item (5-point scale, reverse coded where applicable) after each condition. Both conditions received favorable ratings overall (Cronbach's α = 0.70 (C), 0.75 (TA)). No individual items reached statistical significance (Wilcoxon signed-rank were all $p > 0.05$) which is expected given our sample size. Nevertheless, directional trends favored the TA condition on shared-goal understanding (TA M=4.45 vs. C M=4.18), expressiveness (TA M=4.18 vs. C M=3.82), and collaboration (TA M=3.36 vs. C M=3.00). The collaboration and shared-goal understanding items showed the largest effect sizes (r=0.55 and r=0.48 respectively). suggesting that think-aloud may most strongly influence the felt sense of *working together* even if the effect is not statistically significant at this sample size.

In the TA specific reflection items, most participants agreed that thinking out loud helped them think through the problem (M=3.82, SD=0.75). Opinions were mixed on naturalness (M=3.00, SD=1.10) and self-consciousness (reverse coded, M=3.27, SD=1.19). When asked which version of the system they would prefer using, 8 out of the 11 participants expressed a preference for the TA version. However, many said it was context-dependent, favoring the TA version for creative prototyping or visual projects over large-scale or backend work. Many that preferred the TA version also stated that thinking out loud was unnatural for them and that they might forget to think out loud when using the TA version of the system.

5.2 How Think-Aloud Context Influenced AI Behavior

To understand when and how think-aloud context shaped the AI's responses, we conducted a thematic analysis across all 11 TA session logs. Researchers reviewed context addition logs, chat transcripts, and the preference and intention model TAP maintained during each each session, coding instances where think-aloud context clearly influenced the AI assistant's response. Three codes emerged: **Untyped Code Influence** (the AI assistant makes code changes informed by think-aloud context that was *not* present in the chat message), **Think-Aloud-Based Suggestion** (the AI assistant proposes a feature or direction in a chat response based on context that was only mentioned out loud), and **Intent Capture** (TAP's preference and intention models record goal or preferences the developer never typed).

Untyped Code Influence occurred in 7 out of the 11 sessions, Think-Aloud-Based Suggestions in 8 of 11, and Intent Capture was universal, ranging from 6–45 untyped items per session). Five sessions also included *proactive responses* — instances where TAP's classification agent detected an error or stuck state from think-aloud speech and automatically routed the transcript to the AI assistant, triggering a response or code fixes. Consistent with TAP's conservative threshold for intervention, these were rare (1–2 per session when they occurred).

5.2.1 Illustrative Instances. The following examples illustrate distinct ways in which think-aloud context shaped AI assistant behavior.

Think-aloud context flowing into code: P11 typed a request about "color palette and pop-up experience," but the AI assistant also added a lace trim and gingham background pattern, directions the participant had only confirmed in their think-aloud. TAP's preference model had captured these aesthetic goals from the think-aloud stream and included them in the context bundle sent with the typed message. The participant shortly after confirmed awareness of this during the session: "I remembered the lace trim, which is cool. I don't think I restated that when I typed it." Similarly, P7 sent a chat message saying "add a shuffle button," and the AI assistant also capitalized inconsistent button labels that the participant had only verbally noted were "bugging me."

Verbal frustrations surfacing as suggestions: P9 typed a request to "create a leaderboard," the AI assistant acknowledged that and also suggested fixing an accidental-click UX bug that the participant had only complained about out loud. TAP's preference model had logged this frustration as a dissatisfaction signal, and the AI assistant surfaced it as a suggestion alongside the unrelated chat message.

Proactive interventions from speech: In P6's session, the participant encountered a broken navigation screen and said out loud: "I don't think there's a back to a home page button... Does this next button even work? It's stuck." TAP's classification agent flagged this as an error with immediate urgency and automatically routed the transcript to the AI assistant, which independently pushed a fix without the participant ever typing a bug report.

5.2.2 Condition for Influence. Two participants (P2 and P4) showed no Untyped Code Influence. In these cases, participants typed lengthy or highly specific goal-directed prompts that left little to no room for think-aloud influence. This suggests that think-aloud influence is most visible when participants leave the design space open when working through a task.

5.3 Participant Awareness

We analyzed moments where participants verbalized awareness that the AI assistant was using their think-aloud as context. Five out of 11 participants (P1, P3, P7, P10, P11) had such moments during the TA sessions. Reactions ranged from brief surprise to ("Oh, it caught that. That's crazy." —P10) to reflective appreciation ("I did kinda have a lot of thoughts, so I'm glad that the voice part is

helping to contextualize" —P7). P3 explicitly tested the system's listening by wondering out loud if it had captured a feature idea, then confirmed it by saying "Oh yeah, it heard it. Oh, nice... Yeah, perfect. That's exactly what I wanted." These moments suggest that when participants noticed the think-aloud influence, it reinforced trust and engagement with the system. The absence of such moments in the remaining sessions does not imply that the influence went unnoticed, participants may not have verbalized their awareness of the system listening while thinking out loud. The participants who verbalized awareness were also ones who felt less self-conscious about thinking out loud.

6 Limitations and Future Work

TAP is one approach to capturing a think-aloud stream, but many others exist and are worth studying. One relatively large limitation of TAP is that it does not explicitly elicit think aloud utterances from users—in our study, that task fell to the interviewer. Future work should examine the degree to which think aloud can be elicited at all, and how systems like TAP should respond to those utterances in synchronous real-time.

Our user study also has several limitations. While directional trends and effect sizes are informative, our small sample size (n=12) limits the statistical power of our quantitative results and they should be interpreted with caution. A counterbalancing error resulted in uneven group distribution (4-4-1-2 across task–condition pairings), though condition order remained nearly balanced (5 TA-first, 6 C-first). Both tasks involved creative front-end prototyping in p5.js, a context where think-aloud is naturally expressive and participants noted they might not prefer the Think-aloud system for backend or large-scale projects.

7 Conclusion

In this paper we present TAP, a system that captures developers' think-aloud speech as context for an AI coding assistant. In a within-subjects study with 11 participants, we found that incorporating think-aloud context enabled the AI assistant to make code changes, propose suggestions, and capture preferences and intentions, informed by words that were not present in goal-driven chat prompts. Participants felt a stronger sense of collaboration with the AI assistant when using the think-aloud system, and the majority preferred it for creative prototyping tasks. These findings suggest that think-aloud speech is a promising channel of context for human-AI communication during programming, complementing typed interaction.

Acknowledgments

References

[1] Andrej Karpathy [@karpathy]. 2025. There's a new kind of coding I call "vibe coding", where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good. Also I just talk to Composer with SuperWhisper. https://x.com/karpathy/status/1886192184808149383

[2] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.

[3] Sallyann Bryant, Pablo Romero, and Benedict du Boulay. 2008. Pair programming and the mysterious role of the navigator. *International Journal of Human-Computer Studies* 66, 7 (July 2008), 519–529. doi:10.1016/j.ijhcs.2007.03.005

[4] Jan Chong and Tom Hurlbutt. 2007. The Social Dynamics of Pair Programming. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, Minneapolis, MN, USA, 354–363. doi:10.1109/ICSE.2007.87

[5] K. Anders Ericsson and Herbert A. Simon. 1993. *Protocol Analysis: Verbal Reports as Data*. MIT Press. https://direct.mit.edu/books/monograph/4763/Protocol-AnalysisVerbal-Reports-as-Data

[6] Ruanqianqian Huang, Avery Reyna, Sorin Lerner, Haijun Xia, and Brian Hempel. 2025. Professional Software Developers Don't Vibe, They Control: AI Agent Use for Coding in 2025. doi:10.48550/arXiv.2512.14012 arXiv:2512.14012 [cs].

[7] Majeed Kazemitabaar, Jack Williams, Ian Drosos, Tovi Grossman, Austin Henley, Carina Negreanu, and Advait Sarkar. 2024. Improving Steering and Verification in AI-Assisted Data Analysis with Interactive Task Decomposition. doi:10.1145/3654777.3676345 arXiv:2407.02651 [cs].

[8] Rebecca Krosnick, Fraser Anderson, Justin Matejka, Steve Oney, Walter S. Lasecki, Tovi Grossman, and George Fitzmaurice. 2021. Think-Aloud Computing: Supporting Rich and Low-Effort Knowledge Capture. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. ACM, Yokohama Japan, 1–13. doi:10.1145/3411764.3445066

[9] Stanley Letovsky. 1987. Cognitive processes in program comprehension. *Journal of Systems and Software* 7, 4 (Dec. 1987), 325–339. doi:10.1016/0164-1212(87)90032-X

[10] Michael Xieyang Liu, Advait Sarkar, Carina Negreanu, Benjamin Zorn, Jack Williams, Neil Toronto, and Andrew D Gordon. 2023. "What it wants me to say": Bridging the abstraction gap between end-user programmers and code-generating large language models. 1–31.

[11] Sydney Nguyen, Hannah McLean Babe, Yangtian Zi, Arjun Guha, Carolyn Jane Anderson, and Molly Q Feldman. 2024. How Beginning Programmers and Code LLMs (Mis)read Each Other. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu HI USA, 1–26. doi:10.1145/3613904.3642706

[12] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces (IUI '23)*. Association for Computing Machinery, New York, NY, USA, 491–514. doi:10.1145/3581641.3584037

[13] Advait Sarkar. 2024. Intention Is All You Need. doi:10.48550/arXiv.2410.18851 arXiv:2410.18851 [cs].

[14] Advait Sarkar and Ian Drosos. 2025. Vibe Coding: Programming through Conversation with Artificial Intelligence. arXiv:2506.23253 [cs] doi:10.48550/arXiv.2506.23253

[15] Priyan Vaithilingam, Ian Arawjo, and Elena L. Glassman. 2024. Imagining a Future of Designing with AI: Dynamic Grounding, Constructive Negotiation, and Sustainable Motivation. In *Designing Interactive Systems Conference*. ACM, IT University of Copenhagen Denmark, 289–300. doi:10.1145/3643834.3661525

[16] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. ACM, New Orleans LA USA, 1–7. doi:10.1145/3491101.3519665

[17] L. Williams, R.R. Kessler, W. Cunningham, and R. Jeffries. 2000. Strengthening the case for pair programming. *IEEE Software* 17, 4 (July 2000), 19–25. doi:10.1109/52.854064

[18] J.D. Zamfirescu-Pereira, Eunice Jun, Michael Terry, Qian Yang, and Bjoern Hartmann. 2025. Beyond Code Generation: LLM-supported Exploration of the Program Design Space. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. ACM, Yokohama Japan, 1–17. doi:10.1145/3706598.3714154