

Cracking CodeWhisperer: Analyzing Developers’ Interactions and Patterns During Programming Tasks

Jeena Javahar¹, Tanya Budhrani¹, Manaal Basha¹,

Cleudson R. B. de Souza², Ivan Beschastnikh¹, Gema Rodríguez-Pérez¹,

¹University of British Columbia, ²Federal University of Pará

jeenajavahar@gmail.com, tanya.budhrani@connect.polyu.hk, manaals@student.ubc.ca,
cleudson.desouza@acm.org, bestchai@cs.ubc.ca, gema.rodriiguezperez@ubc.ca

Abstract—The use of AI code-generation tools is becoming increasingly common, making it important to understand how software developers are adopting these tools. In this study, we investigate how developers engage with Amazon’s CodeWhisperer, an LLM-based code-generation tool. We conducted two user studies with two groups of 10 participants each, interacting with CodeWhisperer - the first to understand which interactions were critical to capture and the second to collect low-level interaction data using a custom telemetry plugin. Our mixed-methods analysis identified four behavioral patterns: 1) incremental code refinement, 2) explicit instruction using natural language comments, 3) baseline structuring with model suggestions, and 4) integrative use with external sources. We provide a comprehensive analysis of these patterns.

Index Terms—Code Generation Tools, LLM, Developer Perceptions, Software Development, User Studies

I. INTRODUCTION

Several IDE-based code generation tools have been released in the past few years, such as GitHub’s Copilot [8], Kite [14], Amazon’s Code Whisperer [20], Tabnine [22], and WPCode [28]. Research reveals that being able to achieve their full potential requires a certain level of guidance to ensure that the tool’s output aligns with the user’s goal [21]. Our research aims to explore **how developers engage with an LLM integrated into their IDE, and the patterns that emerge from their interactions across programming tasks.**

To identify key interaction patterns, we conducted an initial study with 10 participants using Amazon CodeWhisperer in their IDEs to solve progressively challenging tasks. Screen recordings from this study provided insights into user behaviors and tool usage. Based on the initial findings from the first study, we developed CodeWatcher [4], a tool that includes a VSCode plugin for capturing and analyzing developer interactions with LLM tools. We then conducted a second user study with 10 new participants using our tool. This provided us with low-level interaction data to study usage patterns.

Our work aims to provide deeper insights into the practical usage of LLM-based tools for code generation by addressing the following research questions:

- RQ1 - How do users interact with and edit code using CodeWhisperer to solve programming tasks, and what detailed interaction patterns emerge?

- RQ2 - What fraction of CodeWhisperer’s suggestions are retained by users?

As part of answering these research questions, we make the following contributions:

- 1) We bridge low-level interaction data with high-level LLM usage by recording and analyzing how users engage with LLMs and how they edit code. This paper is one of the first studies to do so; and
- 2) We identify several new behavioural patterns associated with using LLM-based code generators.

II. RELATED WORK

LLMs are neural networks with billions of parameters pre-trained on large corpora of unlabeled text through supervised learning [9]. When applied to software development, LLMs can generate code from natural language descriptions. In this paper, we use CodeWhisperer [20] now Amazon Q, a tool built by Amazon Web Services [1].

User studies of LLM-based code generators are broadly classified them into two groups: (i) studies of GitHub Copilot, and (ii) studies of “customized” code generation tools, i.e., tools created by the study authors as explained by Mendes et al. [18]. Mendes et al. discussed two customized tools: NL2Code [29], and GenLine [11]. Xu et al. [29] concluded that most NL2Code’s users had either a positive or neutral experience using it. Meanwhile, Jiang et al. [11] discuss an evaluation of GenLine with 14 participants using it for a week: participants felt they had to “learn” how to interact with the tool because of unexpected model responses.

Barke et al. conducted a grounded theory analysis of GitHub’s Copilot being used by 20 participants [2], identifying two primary ways users interact with an LLM-based code generator: acceleration and exploration. Another study worth mentioning is by Vaithilingam et al. [25] who conducted an experiment with 24 students using GitHub Copilot. Similar to Barke et al., students enjoyed using Copilot because it provided code that could be used as a “starting point”, even when this code was incorrect. They also reported two coping strategies to deal with Copilot’s limitations: “to accept the incorrect suggestion and attempt to repair it” or simply stop using the tool. Building on these previous studies, our work records users’ interactions with CodeWhisperer and analyzes

how users edit code. This approach bridges low-level interaction details with high-level LLM usage, offering an in depth view of developer engagement and the practical application of LLM-generated suggestions.

Contrary to some of the previous studies which rely on surveys, video recordings, interviews, etc [2], [18], [19], [24], [25], [26], [29] with a focus on user reactions, our paper uses low level interaction data to identify new behavioral patterns.

To the best of our knowledge, only two papers adopt a similar approach to ours. Ziegler et al. [30], analyzed internal Copilot developer usage data, including the completion shown to each developer, the completion accepted by the developer for inclusion in the source file, the number of characters in an accepted completion, among others. Tang et al. [23] who examined developer behaviour during validation and repair of LLM-generated code using eye-tracking and IDE actions with their tool, CodeGRITS. By using the data captured by our tool-independent plugin [4], we complement the previous studies and provide a new perspective to the study of interaction between developers and LLMs fine-tuned for code generation.

III. METHODOLOGY

Our study was conducted in three phases (See Figure 1). First, we carried out an initial user study using screen recordings to observe how participants interact with CodeWhisperer. This phase resulted in a detailed codebook outlining user interactions with CodeWhisperer. The codebook was used to create an IDE plugin, CodeWatcher [4], that automatically logs users' interactions with the IDE by capturing significant changes in the text body and specific keystrokes. Second, we conducted a second user study, using the same setup as the study in phase 1 using CodeWatcher for recordings. Finally, we analyzed the logs to uncover patterns in user interactions.

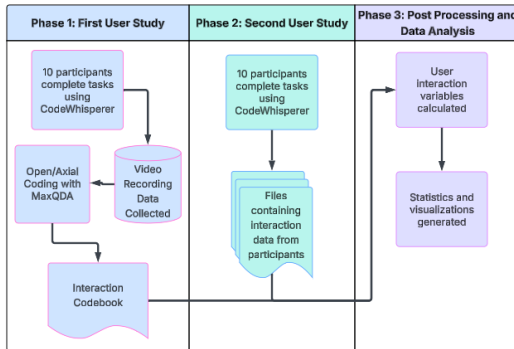


Fig. 1: Study Phases

A. Phase 1: Initial User Study

We recruited 10 undergraduate Computer Science students through posters, mailing lists, and student clubs. A screening survey collected demographics and programming proficiency, and we selected participants to ensure diversity in gender, native language, and experience. The study was approved by our university's ethics board and included five self-identified

men and five self-identified women from varied linguistic and national backgrounds. None of the participants had prior experience with Amazon CodeWhisperer.

Study setup and Data Collection: The user study included a two-hour morning session, a one-hour lunch break, and a two-hour afternoon session. Participants were introduced to LLMs and CodeWhisperer, followed by hands-on Python tasks using VSCode. They worked at their own pace, recorded their screens, and submitted their work anonymously. Compensation included \$25 CAD per session.

Python programming tasks: We selected six real-world Python programming tasks involving basic Python, file manipulation, and operating system problems of varying difficulty. We selected tasks used by Xu et al [29]. Detailed information about the user study and programming tasks can be found in our online appendix [10].

Data Analysis: To analyze screen recordings, we used *open* and *axial* coding, key methods in qualitative analysis [5]. The first author began with *open coding* on three initial recordings (first tasks of three participants), using MaxQDA [17] to segment and label interactions. Each time a new code was identified, recordings were revisited for consistency. Codes were reviewed and refined through discussions with two other authors. This process was repeated for recordings of the second and third tasks, with partial participant overlap. Saturation was reached after the third round. Next, we performed *axial coding*, grouping codes into two categories: *CodeWhisperer Suggestions* (i.e., natural language or code prompts) and *Participant Interactions* (i.e., common user behaviors). Finally, the comprehensive codebook is available in our online appendix [10].

B. The CodeWatcher Plugin

We developed CodeWatcher [4], a VSCode extension built with JavaScript and Node.js, to run alongside CodeWhisperer without impacting its functionality. It logs real-time keystrokes and document changes in JSON format. The plugin tracks eight distinct types of user interactions within the IDE during each coding session, on a per-file basis: Start (when editing begins), End (when editing stops), Insertion (adding text), Deletion (removing text), Focus (when a file becomes active), Unfocus (when a file becomes inactive), Copy (copying text), and Paste (pasting text). For each event, the CodeWatcher captures some of the following four event properties: (1) Type, (2) Time, (3) Text and (4) Line. Type records which of the eight events was captured. Time represents the timestamp of when there was an increase in the document text larger than three characters at a time. Text corresponds to the changed text if applicable. Line is the text of the line where the change occurred if applicable [4].

C. Phase 2: Second User Study

The recruitment strategy for this phase is the same as that of phase one. In this case, the participants included five self-identified men, four self-identified women, and one individual who chose not to disclose their gender identity. All participants

were Computer Science students at our university, consisting of one undergraduate and nine graduate students.

Study setup: The setup for this user study mirrored phase one, except participants used our CodeWatcher plugin instead of screen recording. Participants anonymously submitted their completed code files and the associated logs after finishing the tasks to a cloud service. Each participant also received a remuneration of CAD \$25 at the start of each session.

D. Phase 3: Post Processing and Data Analysis

During post-processing, we quantitatively analyzed the user event logs to identify and categorize the users' interactions with CodeWhisperer. Specifically, to answer RQ1, we defined and computed the user interaction variables from the event logs as reported in the Table I. To address RQ2, we analyzed how much of CodeWhisperer's suggested code was retained by participants. This involved examining *Insertion*, *Copy*, and *Paste* events from CodeWatcher, since only inserted or pasted text could appear in the final code. We categorized events, filtered out invalid or duplicate insertions based on context, and matched valid inserted lines against each participant's final submitted Python file. A detailed description of the line-matching algorithm is provided separately.

IV. RESULTS

Some participants showed no recorded interactions with CodeWatcher, possibly due to not using CodeWhisperer suggestions or logging failures. Our dataset includes 2,527 interactions in Task 1, 2,323 in Task 2, and 4,159 in Task 3

A. RQ1: How do users interact with and edit code using CodeWhisperer to solve programming tasks, and what detailed interaction patterns emerged from that?

Table II shows the user interactions recorded for Task 1 (Basic Python programming tasks). These interactions represent the aggregated user interactions for Task 1.1 and Task 1.2. In these tasks, consecutive single letter deletions (CSLD) were the most frequent interactions, accounting for 29% of all actions, followed by partial generated insertions (PGI) at 20%. Additionally, Task 1 exhibited relatively high rates of focus (F) and unfocus (UF) interactions, constituting 10% and 9% of interactions, respectively, which were notably higher compared to other types of interactions.

For Task 2 (file manipulation programming tasks), the most frequent interactions were PGI (24%) and CSLD (23%), maintaining a similar trend to Task 1. Complete generated insertions (CGI) and complete deletions (CD) were also prevalent, comprising 7% and 11% of all interactions, respectively. Focus and unfocus interactions remained consistent with Task 1, both at 10%. Table III shows the user interactions recorded for Task 2. These interactions represent the aggregated user interactions for Task 2.1 and Task 2.2. In Task 3 (OS programming tasks), PGI remained the most common interaction type at 27%, followed closely by CSLD at 23%. Unfocus interactions (UF) were recorded at 7%, reflecting a moderate level of task engagement within the IDE. Table IV shows the user

interactions recorded for Task 3. These interactions represent the aggregated user interactions for Task 3.1 and Task 3.2.

Meanwhile, the rates of CGI (Completed Generated Insertions) decrease as the task difficulty increases: percentages go from 6.88% (Task 1) to 5.62% (Task 2) to 3.82% (Task 3). This could be explained by the limitations of LLMs in handling more complex programming problems [7].

To examine interaction patterns, we plotted user interaction variables for each task over normalized time, from task start to completion. The x-axis represents normalized time, the y-axis participant IDs, and a legend distinguishes interaction types. The plots are available in the online appendix. In the coding task T1.2, several participants had long gaps between interactions, possibly reflecting a learning curve, uncertainty in using CodeWhisperer, or reliance on personal knowledge for a simpler task. In contrast, the more challenging T3.1 showed denser interaction patterns, likely due to its complexity prompting more frequent tool use. Focus/unfocus events revealed differences in coding style: some participants worked continuously in the IDE, while others switched contexts more often. Overall, higher task difficulty correlated with increased interactions, suggesting both greater reliance on CodeWhisperer and growing familiarity with its ability to reduce cognitive load.

1. Incremental code refinement: partial and complete insertions along with consecutive single letter deletions are the most predominant user interaction variables in the Figures. In other words, it was a common occurrence that participants tended to accept end-of-line suggestions, such as completing a comment or finishing the syntax of a function definition then refining portions of it to their liking.

2. Explicit Instruction Using Natural Language Comments: Participants typically include Natural Language Comments (NLC) containing command words like ``Create," ``Write," and ``Make," to prompt the CodeWhisperer and guide it on what to do. There exists a clear distinction between participants' comments made for themselves and comments made to interact with CodeWhisperer, the former being formatted in an explanatory manner, detailing how their code works, while the latter utilizing action words for commands and requests. For example, these are some of the comments found in Participant 6's final Python file for Task 3.2:

```
# In the process directory or file names containing
  dates are renamed such that the date is
  reformatted from yyyy-mm-dd format to dd-mm-yyyy
  format.
```

Comparatively, these are some of the comments found in the log file that did not make it to the output script but allowed the user to interact with CodeWhisperer:

```
# get the current working directory
[...]
# get the date from the file name
```

3. Base-line structuring with model suggestions: We also observed how participants tend to accept suggestions that they may not particularly need but are syntactically similar to what they were intending to write. For example, participant

TABLE I: User Interaction Variables with Derivation Approach

	Category	Description	Derivation Approach
CGI	Complete generated insertion	CodeWhisperer offers a suggestion before the participant begins writing, and the participant accepts the suggestion in its entirety.	Filter for <i>insertion</i> objects likely to contain text generated by CodeWhisperer, ensure that <i>line</i> contained no additional text preceding <i>insertion</i>
PGI	Partially generated insertion	CodeWhisperer offers a suggestion after the participant begins writing, and the participant accepts the suggestion to finish what they have already written.	Filter for <i>Insertion</i> objects likely to contain text generated by CodeWhisperer, ensure that <i>line</i> contained additional text preceding <i>insertion</i>
CSLD	Consecutive single letter deletion	The participant uses the backspace key to erase a word, letter by letter. If multiple single-letter deletions occur within the same second or within a one-second difference, it is counted as a CSLD.	Group <i>deletion</i> objects where text is one letter with timestamp difference ≤ 1 second
PSLD	Partial single letter deletion	The participant uses the backspace key to erase just one letter.	Isolated single letter deletion object
CD	Complete deletion	The participant highlights and deletes the entire code, leaving nothing but a newline, carriage return, or space.	Filter for deletion objects where line property is empty
PD	Partial deletion	The participant highlights and deletes only a portion of the code, leaving text on the line	Filter for deletion objects where line property is not empty
F	Focus	The participant shifted focus to the IDE/ programming task and CodeWhisperer.	<i>focus</i> object
UF	Unfocus	The participant shifted focus away from the programming task and CodeWhisperer.	<i>unfocus</i> object
ES	External source	The participant utilizes outside sources to add to their code, without the help of CodeWhisperer.	<i>unfocus</i> object followed by a <i>paste</i> object
IS	Internal source	The participant copies and pastes code from within the IDE they are writing in.	<i>copy</i> object followed by a <i>paste</i> and <i>insertion</i> , where text property on <i>insertion</i> object corresponds to text property on <i>copy</i> object
C	Copy	The participant copies a portion of the code internally	<i>copy</i> object

TABLE II: User Interactions during Task 1

User	PGI	CGI	CSLD	PSLD	CD	PD	F	UF	ES	IS	C	Sum
1	6	3	12	3	1	1	6	6	3	0	1	42
2	150	24	105	40	44	34	12	0	7	1	1	418
3	0	0	0	0	0	0	0	0	0	0	0	0
4	15	1	22	5	9	10	11	11	3	0	0	87
5	77	45	272	35	59	40	62	49	0	10	10	659
6	63	46	117	24	32	14	54	59	13	5	11	438
7	84	9	55	10	15	20	21	22	6	0	0	242
8	33	14	59	9	31	24	53	51	6	10	3	5 292
9	54	3	43	10	11	3	26	26	3	2	2	183
10	15	29	36	12	21	14	16	13	6	0	4	166
Sum	497	174	721	148	223	160	261	237	51	21	34	2527
Mean	49.7	17.4	80.11	14.8	22.3	16	26.1	23.7	5.1	2.1	3.77	
Med	43.5	11.5	55	10	18	14	18.5	17.5	4.5	0.5	2	
StDev	46.24	17.72	80.01	13.62	19.14	13.63	22.18	21.99	4.17	3.24	4.17	

TABLE IV: User Interactions Task 3

User	PGI	CGI	CSLD	PSLD	CD	PD	F	UF	ES	IS	C	Sum
1	304	20	312	11	66	160	16	10	9	8	15	931
2	0	0	0	0	0	0	0	0	0	0	0	0
3	28	20	40	4	49	16	18	11	8	0	1	195
4	95	7	92	24	60	80	13	9	4	18	24	426
5	289	53	180	74	104	74	11	6	5	10	11	817
6	155	15	125	35	51	50	53	60	12	3	7	566
7	54	7	39	25	14	35	5	5	1	0	0	185
8	76	6	20	7	15	25	95	101	16	3	9	373
9	106	14	82	38	29	58	74	90	15	6	6	518
10	15	17	57	10	16	17	3	3	9	0	1	148
Sum	1122	159	947	228	404	515	288	295	79	48	74	4159
Mean	112.2	15.9	105.22	22.8	40.4	51.5	28.8	29.5	7.9	4.8	8.22	
Med	85.5	14.5	82	17.5	39	42.5	14.5	9.5	8.5	3	7	
StDev	107.51	14.62	91.97	22.19	31.57	46.20	33.19	38.83	5.46	5.88	7.75	

10 used the CodeWhisperer’s suggestion to write the phrase `''Write a program that prints your favorite color.''` Using a complete generated insertion, the participant accepted the suggestion in its entirety, only to delete more than half of it until only `''Write a program tha` was left. This format allowed them to finish the comment with the command they actually wanted, which was `''Write a program that also in the process directory or file names containing dates are renamed such that the date is reformatted from yyyy-mm-dd format to dd-mm-yyyy format.''`

TABLE III: User Interactions during Task 2

User	PGI	CGI	CSLD	PSLD	CD	PD	F	UF	ES	IS	C	Sum
1	39	23	164	7	30	27	20	20	0	1	4	335
2	238	44	116	42	98	47	27	16	11	0	3	642
3	28	20	40	4	49	16	18	11	8	0	1	195
4	4	0	0	0	0	0	0	0	0	0	0	4
5	0	0	0	0	0	0	0	0	0	0	0	0
6	110	19	22	13	38	53	77	81	12	1	4	430
7	51	10	47	7	8	14	19	13	3	0	0	172
8	3	4	17	3	15	18	38	41	10	2	7	158
9	66	4	96	10	14	14	38	55	8	0	1	306
10	23	6	19	0	10	2	3	3	5	0	0	71
Sum	562	130	521	86	262	191	240	240	57	4	20	2313
Mean	62.44	14.44	65.12	9.55	29.11	21.22	26.66	26.66	6.33	0.44	2.5	
Med	39	10	43.5	7	15	16	20	16	8	0	2	
StDev	73.71	13.76	54.22	12.91	30.22	18.26	23.02	26.97	4.55	0.72	2.44	

TABLE V: Percentage and sum of matched lines for Tasks 1, 2, and 3. NLC means Natural Language Comments

	Tasks 1			Tasks 2			Tasks 3		
	NLC	Code	Sum	NLC	Code	Sum	NLC	Code	Sum
Generated Insertion Line	89	613	702	71	452	523	100	487	587
Lines with match found	34	237	271	41	260	301	88	314	402
Altered/Deleted	55	376	431	30	192	222	12	173	185
Percentage of matches	38%	39%	-	58%	58%	-	88%	65%	-

4. Integrative use with external sources: In Participant 9’s interaction progression (Figure 2b), we see a slew of focus and unfocus interactions. This can mean two things: the user decided to (1) exit the IDE to consult outside sources when the aid of CodeWhisperer was not sufficient, or (2) take a break, read through the code they have so far, and think about how they can proceed, instead of constantly accepting the end-of-line suggestions offered by CodeWhisperer. In Figure 2a, for Task 1, this is a lot more prevalent.

B. RQ2: What fraction of CodeWhisperer’s suggestions are retained by users?

Table V shows the percentage of CodeWhisperer’s suggested NLC and Code that appeared in the final output files submitted by participants during Phase 3 of the second user study. Table V shows a trend of increasing accepted suggestions being incorporated into the final output code over time. In Task 1 we observe a 38% match rate for NLC, indicating that over one-third of CodeWhisperer’s suggested comments were directly incorporated without edits. This could suggest that the participants made use of the NL comments as explicit instructions to prompt CodeWhisperer on code to write, then deleted these after the code was given. Furthermore, the retention rate for *code* lines in Task 1 is 39%.

In Task 2, the match rate for NLC increases significantly to 58%, representing a 52.6% increase compared to Task 1. The match rate for code lines also increased to 58%, reflecting a higher level of reliance on CodeWhisperer’s generated code. Again, this might indicate that study participants were getting more comfortable with the tool and its suggestions as they progressed towards the difficult tasks. In Task 3, the match rate for NLC peaks at 88%, while the match rate for code lines also rises to 66%. These results might indicate that by Task 3, participants had developed a significant level of confidence in CodeWhisperer’s ability to generate code, as reflected by the high proportion of unedited suggestions included in their final submissions.

V. DISCUSSION

Behavioral patterns: Users often find significant value in the convenience and efficiency of AI-assisted coding [2], [3], [6], [13], [16], [18]. Aligned with our findings for RQ1, participants often engaged in an iterative process of accepting and refining CodeWhisperer suggestions, indicating they found value in them. This mirrors prior work [18], [25], where users viewed AI-generated code, even when incorrect, as useful starting points for crafting correct solutions.

A similar approach is observed in the context of comments: users accepted suggestions for comments that were syntactically similar to what they were intending to write. We also observed an integrative use of external sources. That is, there were instances where participants relied on external resources (e.g., Stack Overflow, ChatGPT) when the suggestions provided by CodeWhisperer were perceived as insufficient. Previous studies have shown that users can experience frustration when interacting with code generation tools [15], [18], [25], which may explain a user’s choice to consult an External Source (see Table I).

LLM’s suggested code retention: Our key finding for RQ2 shows that the proportion of AI-generated content retained by users increased over time. Comment retention rose from 38% to 88%, and code retention from 39% to 65%. This may be due to two factors: the increasing difficulty of tasks encouraging participants to accept more suggestions, and growing familiarity with the tool, leading to greater trust in its output. Moreover, the reliance on CodeWhisperer might allow

participants to offload routine work and focus on higher-level problem solving, using the tool to reduce cognitive load.

Implications for Tool Builders Our four observed patterns: (1) incremental code refinement, (2) use of natural language comments, (3) baseline structuring with suggestions, and (4) integrative use of external sources; offer key insights for improving LLM-based coding tools.

The first and third patterns align with prior work [2], [18], highlighting that both code and comment suggestions serve as valuable starting points. Tools could better support this by allowing users to incrementally accept suggestions, enabling easier customization, a feature that may require IDE redesign [27]. In the second pattern, participants used comments for both self-documentation and prompting the LLM. Capturing these guiding comments could help preserve the design rationale behind code generation. The fourth pattern emphasizes the need for external validation. As Kabir et al. [12] found, users overlooked inaccuracies in ChatGPT’s responses 39% of the time. Tool builders should consider integrating LLMs with trusted search or documentation sources to offer more accurate, context-aware support when AI alone is insufficient.

Threats to Validity: Our small sample limits our ability to distinguish the effects of demographics versus individual coding styles, so larger cohorts are needed for generalizability. CodeWatcher logs document changes and keystrokes, then infers AI-generated code. This post-processing, combined with our small sample, risks invalid or outlier observations. Finally, both participants familiar with the tool and those unaware of its functionality could skew results (e.g., missing interactions in Figures 2a and 2b may reflect logging failures).

VI. CONCLUSION AND FUTURE WORK

This paper examines user interactions with LLM-based code generators, focusing on behavior, trust, and perceptions during programming tasks. We investigated how users engage with these tools, how their trust evolves, and how they perceive benefits in productivity, efficiency, and learning.

Future work should explore the long-term impact of LLM use on skill development, particularly the potential erosion of foundational coding skills among novices. Studying usage patterns and trust across diverse demographic groups with larger samples could also offer valuable insights. Understanding these long-term effects and optimizing tool design for both learning and productivity remains a key research priority.

ACKNOWLEDGMENT

We thank GOOGLECA (PWPU GR028067) and CNPq (420406/2023-9 and 442779/2023-2) for funding this research.

REFERENCES

- [1] *CodeWhisperer*, 2024. available at <https://docs.aws.amazon.com/codewhisperer/latest/userguide/whisper-line-by-line-1.html>.
- [2] BARKE, S., JAMES, M., AND POLIKARPOVA, N. Grounded copilot: How programmers interact with code-generating models. *Association for Computing Machinery* 7 (April 2023).

- [3] BASHA, M., AND RODRÍGUEZ-PÉREZ, G. Trust, transparency, and adoption in generative ai for software engineering: insights from twitter discourse. *Information and Software Technology* (2025), 107804.
- [4] BASHA, M. R., RIBEIRO, A. M., JAVAHAR, J., RODRÍGUEZ-PÉREZ, G., AND DE SOUZA, C. R. B. Codewatcher: Ide telemetry data extraction tool for understanding coding interactions with llms. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)* (2025), ICSME'25, IEEE.
- [5] CORBIN, J., AND STRAUSS, A. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. SAGE Publications, Inc., Thousand Oaks, California, 2015. Fourth Edition.
- [6] CUI, Z., DEMIRER, M., JAFFE, S., MUSOLFF, L., PENG, S., AND SALZ, T. The Effects of Generative AI on High Skilled Work: Evidence from Three Field Experiments with Software Developers. *SSRN eLibrary* (2024).
- [7] DOU, S., JIA, H., WU, S., ZHENG, H., ZHOU, W., WU, M., CHAI, M., FAN, J., HUANG, C., TAO, Y., LIU, Y., ZHOU, E., ZHANG, M., ZHOU, Y., WU, Y., ZHENG, R., WEN, M., WENG, R., WANG, J., CAI, X., GUI, T., QIU, X., ZHANG, Q., AND HUANG, X. What's wrong with your code generated by large language models? an extensive study, 2024.
- [8] GITHUB. Github copilot: Your ai pair programmer, 2024. available at <https://github.com/features/copilot>.
- [9] HUMZA, N., ASAD, U. K., SHI, Q., MUHAMMAD, S., SAEED, A., MUHAMMAD, U., NAVEED, A., NICK, B., AND AJMAL, M. A comprehensive overview of large language models. *Elsevier* (April 2024).
- [10] JAVAHAR, J., BUDHRANI, T., BASHA, M., DE SOUZA, C. R. B., AND RODRÍGUEZ-PÉREZ, G. Cracking codewhisperer: Analyzing developers interactions and patterns during programming tasks, Aug 2025. available at <https://osf.io/e9xvy/>.
- [11] JIANG, E., TOH, E., MOLINA, A., OLSON, K., KAYACIK, C., DONS-BACH, A., CAI, C. J., AND TERRY, M. Discovering the syntax and strategies of natural language programming with generative language models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (2022), pp. 1–19.
- [12] KABIR, S., UDO-IMEH, D. N., KOU, B., AND ZHANG, T. Is stack overflow obsolete? an empirical study of the characteristics of chatgpt answers to stack overflow questions. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2024), CHI '24, Association for Computing Machinery.
- [13] KAZEMITABAAR, M., HOU, X., HENLEY, A., ERICSON, B., WEINTROP, D., AND GROSSMAN, T. How novices use llm-based code generators to solve cs1 coding tasks in a self-paced learning environment. *Association for Computing Machinery* (2023).
- [14] KITE. Kite blog - code better python - content for advanced & beginner devs, 2019. available at <https://www.kite.com/blog/>.
- [15] LIANG, J. T., YANG, C., AND MYERS, B. A. A large-scale survey on the usability of ai programming assistants: Successes and challenges. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering* (2024), pp. 1–13.
- [16] LIAO, V., AND SUNDAR, S. Designing for responsible trust in ai systems: A communication perspective. *FACCT 2022* (June 2022).
- [17] *Qualitative Analysis Software — Powerful and Easy-to-use*, 2023. <https://www.maxqda.com/qualitative-data-analysis-software>.
- [18] MENDES, W., SOUZA, S., AND DE SOUZA, C. “you’re on a bicycle with a little motor”: Benefits and challenges of using ai code assistants. In *Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects of Software Engineering* (New York, NY, USA, 2024), CHASE '24, Association for Computing Machinery, p. 144–152.
- [19] MURALI, V., MADDILA, C., AHMAD, I., BOLIN, M., CHENG, D., GHORBANI, N., FERNANDEZ, R., NAGAPPAN, N., AND RIGBY, P. C. Ai-assisted code authoring at scale: Fine-tuning, deploying, and mixed methods evaluation. *Proc. ACM Softw. Eng.* 1, FSE (jul 2024).
- [20] SERVICES, A. W. Ai coding assistant - amazon developer, 2022. available at <https://aws.amazon.com/q/developer/>.
- [21] SUBRAMONYAM, H., PEA, R., PONDOC, C., AGRAWALA, M., AND SEIFERT, C. Bridging the gulf of envisioning: Cognitive challenges in prompt based interactions with llms. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2024), CHI '24, Association for Computing Machinery.
- [22] TABNINE. Tabnine ai code assistant, 2024. available at <https://www.tabnine.com/>.
- [23] TANG, N., AN, J., CHEN, M., BANSAL, A., HUANG, Y., MCMILLAN, C., AND LI, T. J.-J. Codegrits: A research toolkit for developer behavior and eye tracking in ide. In *46th International Conference on Software Engineering Companion (ICSE-Companion '24)* (2024), ACM.
- [24] TANG, N., CHEN, M., NING, Z., BANSAL, A., HUANG, Y., MCMILLAN, C., AND LI, T. J.-J. A study on developer behaviors for validating and repairing llm-generated code using eye tracking and ide actions, 2024.
- [25] VAITHILINGAM, P., ZHANG, T., AND GLASSMAN, E. L. 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* (2022), pp. 1–7.
- [26] WANG, R., CHENG, R., FORD, D., AND ZIMMERMANN, T. Investigating and designing for trust in ai-powered code generation tools. In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency* (New York, NY, USA, 2024), FACCT '24, Association for Computing Machinery, p. 1475–1493.
- [27] WANG, R., CHENG, R., FORD, D., AND ZIMMERMANN, T. Investigating and designing for trust in ai-powered code generation tools. In *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency* (New York, NY, USA, 2024), FACCT '24, Association for Computing Machinery, p. 1475–1493.
- [28] WPCODEBOX. Wordpress code snippets with cloud, 2024. available at <https://wpcodebox.com/>.
- [29] XU, F. F., VASILESCU, B., AND NEUBIG, G. In-ide code generation from natural language: Promise and challenges. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 2 (2022), 1–47.
- [30] ZIEGLER, A., KALLIAMVAKOU, E., LI, X. A., RICE, A., RIFKIN, D., SIMISTER, S., SITTAMPALAM, G., AND AFTANDILIAN, E. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming* (2022), pp. 21–29.

APPENDIX

TABLE VI: CodeBook of Events

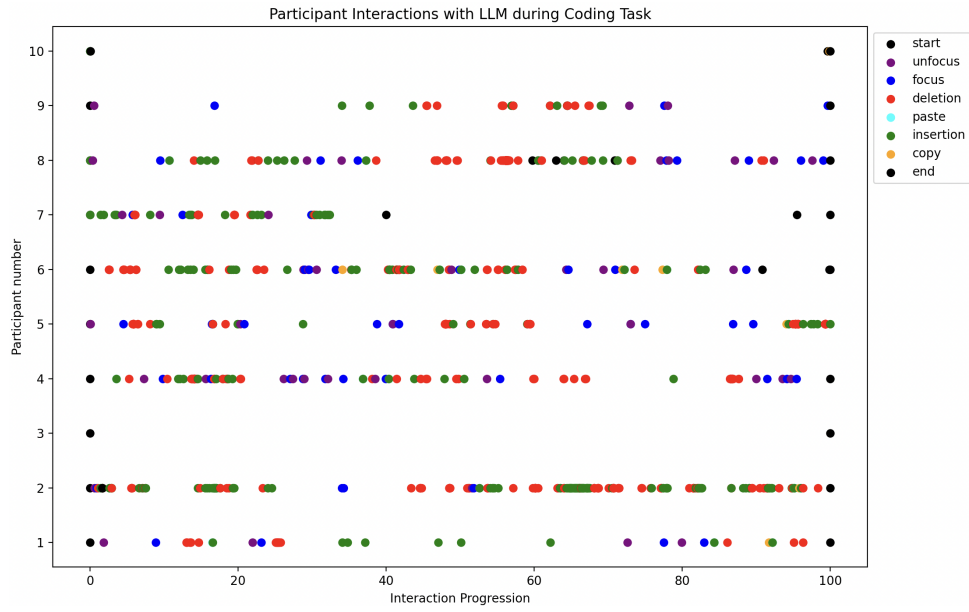
Event	Description
CodeWhisperer Events	
PromptSuggestion	CodeWhisperer makes a suggestion to complete the comment the user is currently typing
CodeSuggestion	CodeWhisperer suggests a code block
PromptCodeSuggestion	CodeWhisperer makes a suggestion to complete the comment the user is currently typing and a corresponding code block
Participant Events	
PromptCreation	Participant begins writing a prompt
PromptCompletion	Participant completes the prompt themselves
PromptSuggestionAcceptExplicit	Participant accepts a CodeWhisperer prompt suggestion in its entirety
PromptSuggestionAcceptImplicit	Participant manually types prompt similar to CodeWhisperer's suggestion
PromptEdit ^{VI,VI}	Participant edits a prompt
PromptSuggestionEdit	Participant edits an accepted prompt
PromptSuggestionDelete	Participant deletes an accepted prompt suggestion
PromptDeletion	Participant deletes a prompt in its entirety
CodeCreation	Participant begins writing code
CodeCompletion	Participant completes a stream of code themselves
CodeDeletion	Participant deletes a stream of code they wrote themselves
CodeSuggestionAcceptExplicit	Participant accepts a CodeWhisperer code suggestion in its entirety
CodeSuggestionAcceptImplicit	Participant manually types code similar to CodeWhisperer's suggestion
CodeEdit	Participant edits a code section untouched by generated code
CodeSuggestionEdit ^{VI}	Participant edits an accepted code suggestion
CodeSuggestionDelete	Participant deletes an accepted code suggestion
CodeSuggestionPartialDelete	Participant deletes parts of a generated code block
SuggestionInduction	CodeWhisperer fails to make a suggestion, making the participant go back and forth to induce a suggestion
ProgramRunSuccessful	The participant runs the program successfully, with expected output
ProgramRunUnsuccessful	The participant runs the program unsuccessfully, with unexpected output
PromptSuggestionExplore	The participant cycles through the prompt suggestions
CodeSuggestionExplore	The participant cycles through code suggestions
UnusualActivity	Participant performs an action prohibited in the experiment (e.g., using other code-generation tools, pasting from the repository)

^{VI} An edit is considered continuous alteration until the cursor moves away from the section being edited.

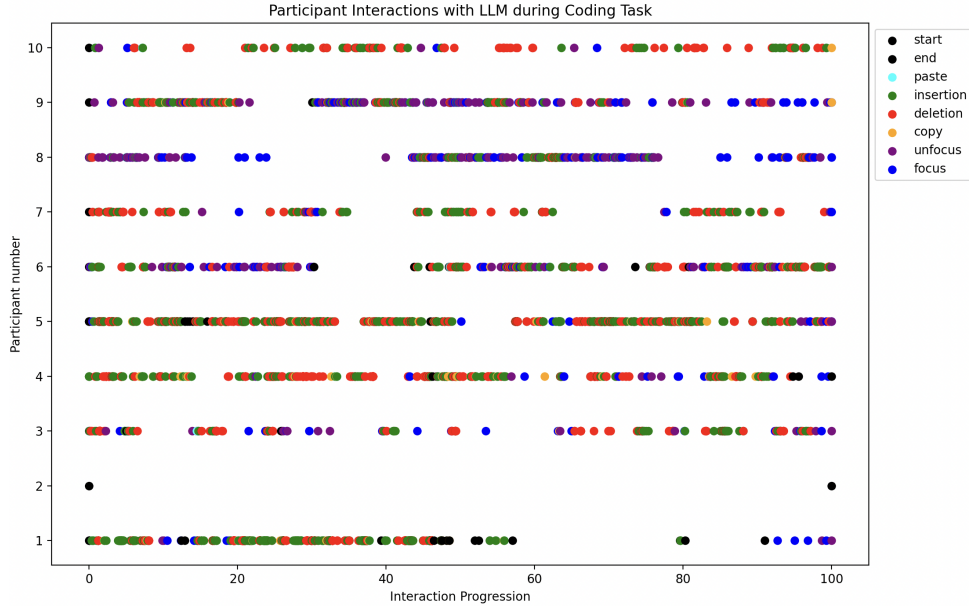
^{VI} Partial deletion is considered an edit for prompts.

Task	Description	Example Output
T1-1	After running <code>python3 T1-1.py</code> , the program randomly generates 100 lower-cased characters (a–z) and 100 integers (1–20 inclusive). These are paired into a dictionary where the key is the character and the value is the sorted list of integers associated with it. The dictionary is printed one key per line, sorted alphabetically.	<pre>\$ python3 T1-1.py a 3 5 6 b 1 1 2 4 5 c 19 ... z 1 12 20</pre>
T1-2	After running <code>python3 T1-2.py</code> , the program prints the date and time one week from now in GMT timezone in <code>mm-dd-yyyy hh:mm (24hr)</code> format.	<pre>\$ python3 T1-2.py 04-23-2020 16:33</pre>
T2-1	After running <code>python3 T2-1.py</code> , the program deletes the first and last columns from <code>data.csv</code> and saves the result to <code>output/output.csv</code> .	<pre>\$ python3 main.py \$ less output/output.csv first_name,last_name,email,gender Merry,MacKetrick,mmackettrick0@latimes.com,Male Calla,Truin,ctruin1@surveymonkey.com,Female</pre>
T2-2	After running <code>python3 T2-2.py</code> , trims leading/trailing whitespaces and blank lines for all text files in <code>data/</code> , normalizes newlines to LF, and converts ISO-8859-15 text files to UTF-8. Non-text files are unchanged. Output saved to <code>output/</code> .	<pre>\$ python3 main.py \$ ls output aaa.txt bbb.txt ccc.txt ddd.png \$ file bbb.txt UTF-8 Unicode text</pre>
T3-1	After running <code>python3 T3-1.py</code> , copies all files/dirs under <code>data/</code> to <code>output/</code> , renaming any dates in filenames from <code>yyyy-mm-dd</code> to <code>dd-mm-yyyy</code> format.	<pre>\$ python3 T3-1.py \$ ls data Photos_2019-03-22 ccc.txt data-02-01-1994.txt 'mybook at 2020-04-01.txt' \$ ls output Photos_22-03-2019 ccc.txt data-02-01-1994.txt 'mybook at 01-04-2020.txt'</pre>
T3-2	After running <code>python3 T3-2.py</code> , processes subdirectories of <code>data/</code> : – For dirs with <code>.txt</code> files: concatenate them (alphanumeric order) into one <code>.txt</code> file named after the dir. – For dirs with <code>.json</code> files: merge lists, re-number id fields starting from 1, save as <code>.json</code> . – Ignore other dirs.	<pre>\$ python3 T3-2.py \$ ls data filelist roster todo \$ ls output filelist.txt roster.json</pre>

TABLE VII: Task Descriptions and Example Outputs



(a) Participant Interactions during Coding Task 1.1



(b) Participant Interactions during Coding Task 3.1

Fig. 2: Comparison of participant interactions across two tasks. To simplify the Figure, we aggregated related interaction variables. For instance, CGI and PGI are presented only as “insertions”.