User Misconceptions of LLM-Based Conversational Programming Assistants

Gabrielle O'Brien
University of Michigan
Ann Arbor, Michigan, USA
elleobri@umich.edu

Antonio Pedro Santos Alves
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, Brazil
apsalves@inf.puc-rio.br

Sebastian Baltes
Heidelberg University
Heidelberg, Germany
sebastian.baltes@uni-heidelberg.de

Grischa Liebel
Reykjavik University
Reykjavik, Iceland
grischal@ru.is

Mircea Lungu
IT University of Copenhagen
Copenhagen, Denmark
mlun@itu.dk

Marcos Kalinowski
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, Brazil
kalinowski@inf.puc-rio.br

Abstract

Programming assistants powered by large language models (LLMs) have become widely available, with conversational assistants like ChatGPT particularly accessible to novice programmers. However, varied tool capabilities and inconsistent availability of extensions (web search, code execution, retrieval-augmented generation) create opportunities for user misconceptions that may lead to over-reliance, unproductive practices, or insufficient quality control. We characterize misconceptions that users of conversational LLM-based assistants may have in programming contexts through a two-phase approach: first brainstorming and cataloging potential misconceptions, then conducting qualitative analysis of Python-programming conversations from the WildChat dataset. We find evidence that users have misplaced expectations about features like web access, code execution, and non-text outputs. We also note the potential for deeper conceptual issues around information requirements for debugging, validation, and optimization. Our findings reinforce the need for LLM-based tools to more clearly communicate their capabilities to users and empirically ground aspects that require clarification in programming contexts.

CCS Concepts

• **Human-centered computing → Natural language interfaces**.

Keywords

generative AI, program synthesis, programmers, large language models, chat, misconceptions, mental model

ACM Reference Format:
Gabrielle O'Brien, Antonio Pedro Santos Alves, Sebastian Baltes, Grischa Liebel, Mircea Lungu, and Marcos Kalinowski. 2025. User Misconceptions of LLM-Based Conversational Programming Assistants. In *Proceedings of* . ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

,

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Programming assistants powered by large language models (LLMs) have rapidly proliferated across software development practices. Several studies indicate their potential to improve developer productivity [49, 53], though this finding is far from universal [5, 45]. Users employ these tools for diverse programming tasks, including information retrieval, code generation, debugging, documentation writing, and learning new technologies [29]. The landscape of available tools is complex and growing: interfaces range from auto-complete engines embedded in development environments to chatbot interfaces that mimic natural language conversations [47], with the latter proving particularly accessible to programmers across skill levels.

A key risk for all users, especially novice programmers, is over-reliance, potentially leading to unproductive practices or insufficient quality control for generated programs [4, 8, 48]. For example, some programmers rely on asking the model to explain its code choices as their entire validation procedure [12, 36], even though LLM-generated explanations may be incorrect [19, 42]—even about code the LLM has just produced. In some contexts, users may not validate outputs at all [12, 21, 39, 40], particularly when confident in a model's abilities [27], or in its capability of providing flawless code [21].

How programmers perceive the abilities and mechanisms behind LLM-based programming tools likely determines their confidence in these tools, a phenomenon that extends beyond programming to other AI domains [3, 6, 22]. This includes forming mental models of tool capabilities and limitations [40]. For example, a user may trust generated answers about a programming library more if they believe a tool can search official documentation rather than relying on a frozen training dataset. Similarly, if a user expects a tool to execute tests, they may be more confident in accepting generated programs. However, these capabilities can differ widely across tools with similar interfaces. In the present study, we investigate what users of conversational LLM-based assistants (like ChatGPT [47]) may misunderstand about how these tools work.

We use a two-phase approach: first, we conduct a brainstorming activity, drawing on existing literature and our experiences teaching programmers to catalog and thematically organize potential misconceptions. We categorize these into misconceptions about the features of specific tools versus misconceptions about LLMs. Second, we perform qualitative analysis of 500 Python programming-related conversations from the openly available WildChat dataset to identify which misconceptions appear in user logs.

We find evidence of misconceptions about the availability of features such as web access and code execution, knowledge cut-offs, non-text outputs, and local machine access. During certain programming activities, particularly debugging, we also observe prompt strategies that may reflect conceptual issues about what runtime and environment information is needed to diagnose and repair programs. We conclude by discussing implications for designing LLM-based tools that better communicate their capabilities to users.

2 Related Work

There is a well-documented tendency towards over-trust in automated systems ("automation bias") both within and beyond programming contexts [2, 16, 52]. Even before the LLM era, work on spreadsheet programming and automated assessments found that less experienced programmers tend to be uncritical of automated feedback [23]. Similarly, user studies with program synthesizers indicate a propensity for overconfidence, termed the "user-synthesizer gap" [14, 18]. Indeed, programmers who understand what tasks a synthesizer can accomplish know when to invoke it rather than try alternative approaches [14]. However, those with poor mental models—particularly those overestimating capabilities—may waste time trying to accomplish impossible tasks. Correspondingly, studies of AI-aided decision making find that users with more accurate mental models achieve more successful outcomes [3, 15, 25].

Previous user-interaction literature suggests that neural-network-based tools pose exceptionally high barriers for developing working mental models. Factors include opaque response rules, high sensitivity to prompt wording changes, and dependency on training data that may be obscure to users [15, 46, 50]. These challenges are compounded by LLMs' tendency to misrepresent their own processes—as one recent analysis noted, "we found that o3 frequently fabricates actions it took to fulfill user requests, and elaborately justifies the fabrications when confronted by the user" [9].

Consistent with this, when exploring mental models of an agentic LLM-based chatbot for information seeking in non-programming contexts, users commonly expected rigid rule-based behavior (e.g., decision trees or database lookups) rather than probabilistic inference for tool calling [6]. Users also expected responses to reflect information retrieval actions rather than training data. Therefore, understanding information provenance was closely related to trust: users wanted to know sources before trusting responses in high-stakes situations [6].

The Computer Science education literature provides additional perspective on how programmers, especially novices, can misconceive LLM-based tools. They defined misconceptions as "understandings that are deficient or inadequate for many practical programming contexts." Sorva [44] Programmer misconceptions about computers, languages, and complex systems have been studied both as targets for educational intervention [17, 37, 41] and to understand barriers to student achievement.

While the existing literature already explores automation bias and weak mental models that lead to miscalibrated trust, there is limited insight into the misconceptions developers form when using conversational LLM assistants for programming in real-world scenarios. By synthesizing prior findings and empirically examining real user conversations, we find evidence of how these misconceptions manifest across chatbot features, and how they might be actionable for tool design and practice.

3 Research Method

We aim to better understand the misconceptions that programmers may have about LLM-based tools. We begin by articulating our assumptions and approach.

3.1 Defining Misconceptions

We define user "misconceptions" as instances where a user erroneously believes a tool has an affordance it lacks (or fails to perceive a critical property). In tool design, affordances are "those fundamental properties that determine just how the thing could possibly be used" [34]. Affordances may be real or erroneously perceived if a user's mental model is incomplete or faulty [35]. For example, a user prompting a conversational assistant might perceive the tool has a web plugin that retrieves information from official documentation before generating responses. If the tool lacked web search capabilities, we would consider this a misconception. This would be relevant to the programmer's confidence in the LLM, as they may weight certainty differently if they believe the response summarizes recent official documentation rather than relying solely on frozen training data.

Misconceptions are context-specific because conversational assistant affordances vary across providers and versions. For example, OpenAI released a Python interpreter for GPT-4 in 2023, initially as an opt-in feature for Plus subscribers, later transitioning to a default feature. Thus, expecting a code interpreter may or may not be a misconception depending on which tool is used, when, and at what subscription tier. A misconception could also reflect insensitivity to a tool property. For example, a user might not perceive that a given LLM-based tool is non-deterministic, becoming confused when the same prompt yields different responses.

3.2 Study Design

Our study involved two activities. First, we conducted a brainstorming activity to identify potential misconceptions in LLM-programmer interactions, drawing on existing literature and our experiences with programmers. This was necessary because we found few studies directly targeting programmers' misconceptions around LLM tools—typically encountering only brief mentions in user interaction studies. Second, we performed qualitative analysis using the openly available WildChat dataset [51] of user interactions with GPT models through a conversational interface. Using conversation logs featuring Python code snippets, we identify which misconceptions appear and in what programming contexts. By focusing on a conversational assistant [47] rather than developer tools like

GitHub Copilot, we expect conversations may skew towards *vibe coding* interactions.

"Vibe coding" describes users relying heavily on natural language to specify programming requirements, often trading quick prototyping for a detailed understanding of their code base [12, 20, 38]. These users may be less experienced, as conversational assistants are often viewed as especially accessible to beginners [12, 47]. However, because WildChat is anonymous, we cannot confirm users' professions or backgrounds. We therefore document what programming activities appear in the dataset.

3.3 Methodological Limitations

Our approach is indirect because we cannot directly measure whether users understand a conversational assistant's affordances. It is also impossible to be certain what misconceptions users have from conversational logs alone, as users might experiment with impossible prompts while exploring the tool. Therefore, we can only identify *potential misconceptions*—interactions where prompts indicate possible misconceptions (e.g., providing a URL to a model without browser access). For brevity, we sometimes use "misconception" in our results, but this should be understood as inference rather than measurement. Acknowledging this limitation, we believe there is value in cataloging potential misconceptions users may have and the programming contexts where they arise.

4 Misconceptions Brainstorming

This brainstorming began as a collaborative activity at the Copenhagen Symposium on Human-Centered AI Adoption in Software Engineering (November 2024), an international workshop supported by the Carlsberg Foundation and Alfred P. Sloan Foundation. Approximately 45 software engineering researchers from institutions across North America, Europe, South America, and Oceania participated, encompassing mid-career to senior experts and representing diverse areas of expertise, including developer productivity, AI tool adoption, and human factors in computing.

The brainstorming activity employed Liberating Structures facilitation techniques to enable broad participation [31]. Through structured small-group discussions, participants shared observations from their research and teaching experiences with programmers using LLM-based tools, identifying recurring patterns of user confusion and misconceptions.

After the workshop, the author team continued working asynchronously to systematize the brainstormed misconceptions, drawing from both the workshop discussions and peer-reviewed literature we had read (continually incorporating new literature into our scheme as we encountered it). Rather than relying strictly on a literature review approach, we chose to incorporate workshop discussion because it contains rich anecdotes deriving from participants' lived experiences in software engineering (all anecdotes are marked as such in our presentation of findings).

We organized the misconceptions into major themes through an iterative process of grouping similar concepts and refining category definitions. Through our discussions, we observed that some misconceptions are about properties of LLMs, whereas others are about properties of specific tools that involve LLM components. We therefore present misconception themes following this distinction.

This exercise is in no way exhaustive, and we do not intend it to be: the set of potential misconceptions is infinitely large, but not all are necessarily interesting. We focused, as in the definition of programmer's misconceptions, on beliefs that we would expect to impact the choices a programmer would make about how to use an LLM-based assistant [44].

4.1 Tool-Specific Misconceptions

Information retrieval mechanism. Users may struggle to form mental models about how LLMs store, retrieve, and use information. Several studies found programmers using conversational LLMs believed them to work via a database [6, 13, 33, 36]. Indeed, information retrieval was the most common mental model students had of ChatGPT-like assistants, with students often expecting keyword-based lookup [33]. However, some tools with LLM components *also* include database components, as in retrieval-augmented generation (RAG) architectures. We expect there to be some user misconceptions about whether a given tool uses RAG before generating responses or relies solely on training data (a confusion observed in non-programming contexts [6]). Relatedly, users might have confusion about *which* knowledge bases are available to a RAG system. One author encountered a colleague expecting that an employer-provided LLM tool had access to all organizational data sources, when this could not be confirmed.

Agentic actions. Given the well-documented tendency to anthropomorphize conversational interfaces [39], we expect users commonly ascribe agentic capabilities to tools lacking these affordances. For example, users might expect LLM tools to conduct web searches or execute programs by default, or that GitHub Copilot would proactively validate a user program's correctness [39]. Other cases include programmers describing ChatGPT as conducting Google searches despite user logs confirming web search was disabled, or testing generated code before returning it [36].

Session memory. In most conversational LLM tools, conversations occur in discrete sessions, potentially confusing users about when information from previous conversations is available. In other words, users, especially non-programmers, might have misconceptions about how code versions are tracked *within* a session [50**?**]. In multi-turn interactions where users explore alternative specifications or debugging repairs, users might expect tools to store "checkpoints" of code as version control, accessible by prompting the tool to return to a previous state.

Session persistence. Users might have misconceptions about whether tools can continue "processing" after a conversation ends. One author shared an experience in which a conversational tool told a user that a task would take several weeks, and the user returned weeks later asking for a progress update.

Scope of access. Users of tools that integrate into development environments or desktop apps may have misconceptions about which local machine data the tool can access. For example, GitHub Copilot users may be unsure which files and directories are "available" to Copilot. Users may also have unexpected beliefs about how to

hide information from a tool, such as one participant's misconception that Copilot ignores in-line code comments when generating suggestions.

Continuous training. LLMs typically have a "knowledge cutoff" corresponding to when training data was collected. Users may have misconceptions about when this occurred for a given tool, or that a cutoff exists at all. This affects reliability for questions about software libraries that evolve after the cutoff. Users might also expect tools "learn" from their coding habits or feedback during a session when no such mechanism exists [50].

Deterministic behavior. Although LLMs may behave deterministically when random seeds are fixed, many user-facing tools don't allow control of such hyperparameters. Users may not understand that many LLM tools won't always give the same response to identical prompts. Several studies reported this as confusing and that LLM unpredictability could challenge learners [26]. In a related study [33], students were "alarmed to find that resubmitting the same prompt could generate different programs."

Model family. While we focus on LLM-based conversational assistants, we note that users could fundamentally misconceive *which* conversational interfaces use LLMs to produce responses. Users might encounter tools with interfaces visually indistinguishable from LLM-based tools that actually run rule-based systems [6, 7, 10].

4.2 LLM Misconceptions

Stability of results. LLM outputs are sensitive to perceptually small changes to prompt authors, like adding whitespace or distractor sentences [32]. This is challenging for users to grasp, particularly because users may not experiment much with prompt variations before forming judgments about model capabilities [50]. Failure to anticipate effects of minor, semantically meaningless modifications has been observed in studies of GitHub Copilot [18] and conversational tools [50].

Groundedness. Users may expect circumstances where LLM hallucinations are impossible, when this cannot be guaranteed. For example, one author encountered a colleague who believed that uploading a data file to a university-provided conversational LLM would prevent hallucinations about that data. Users might also expect LLMs cannot hallucinate about meta-information, such as responding to "Which model are you?" If users think of LLMs as translators between natural language and code, this may parallel expecting certain "system commands" exist.

Native explainability. Users may expect LLMs cannot hallucinate when answering questions about code they generated in the same conversation, when this is untrue [28]. This misconception may contribute to users asking LLMs to justify code suggestions as verification [14]. One author observed a developer expecting "native explainability": asking an LLM to explain why it suggested certain code, confident the answer would faithfully represent the model's reasoning process. The user didn't discern that while explanations might "make sense," they don't reflect explainable AI capabilities regarding previously generated code.

Symbolic logic. Users may be confused that LLMs sometimes respond to prompts, including algebra or logic problems, with correct answers yet don't arrive at such responses via the same mechanism as a calculator (and have many unintuitive sensitivities to how math problems are presented [32]).

Context window. There is a "monotonicity belief" where users believe that giving more information to code synthesizers would always improve performance [18] . In fact, managing the context window is a meta-cognitive challenge [46], especially given highly non-linear relationships between LLM recall of information and its position in the context window [30]. Context window size also differs across models and tools, challenging users to track this information.

4.3 The WildChat Dataset

A challenge for studying how users interact with conversational LLM-based tools in realistic settings is that many such datasets are proprietary. We selected the openly available WildChat corpus of over 1 million conversations between anonymous users and a free, publicly available chatbot [51]. We also considered LMSYS-Chat-1M, but this dataset emphasized a "gamified" interface inviting head-to-head comparisons of 25 LLM models. We selected WildChat because its user interface seemed more ecologically valid for our purposes. WildChat data was collected between April 2023 and May 2024, with 2,713,695 turns in 1,039,785 conversations. Users interacted with a chat interface hosted on Hugging Face[1] supported by OpenAI's GPT-3.5-Turbo and GPT-4 APIs. As a mandatory condition to access the chatbot, users agreed to share chat transcripts, IP addresses, and request headers. Users who affirmatively consented to data sharing could access the service free of charge. Critically, the API calls give us complete visibility into enabled plugins. These API calls did not allow the chatbot to support web search, file uploads, non-text inputs/outputs, or "tool calling" (such as code interpreters). Each conversation was a distinct session; users could not create accounts, so the chatbot had no "memory" of previous conversations. Additionally, the models were text-only. Based on IP addresses, users were located predominantly in the US (21.60%), Russia (15.55%), and China (10.02%). Analysis of 1,000 random prompts indicated 6.7% were coding-related (other categories: assisting/creative writing 61.9%, analysis/decision explanation 13.6%, factual info 6.3%). English was the most common prompt language (53%), followed by Chinese (13%) and Russian (12%).

4.3.1 Pre-processing. To understand user interactions in programming contexts, we focused on coding-related conversations. Although WildChat authors estimated the fraction of coding-related prompts, they did not publish topic labels for specific conversations. We created a pipeline to filter relevant conversations. First, we identified coding-related conversations by applying a regular expression filter for code block formatting characters (e.g.,) in user prompts and chatbot responses. When GPT models respond with code formatting, the formatting string typically includes a language tag (such as `python```). We estimated the number of conversations containing various popular programming languages

[1] https://huggingface.co/spaces

from Stack Overflow's 2025 developer survey [1]. Out of 65,590 conversations with code block formatting, Python was most common (23,353), followed by Java (11,852), SQL (11,852), and JavaScript (7,312). Since all authors were familiar with Python and it was most common, we restricted our analysis to this subset. We also filtered for English-language conversations (Figure **??**b) using WildChat's machine-generated language labels. This left 11,418 candidate conversations, from which we selected 500 at random for qualitative analysis.

4.3.2 Annotation tool. We created a custom annotation tool to visualize turn-by-turn interactions between users and model responses (Figure 1). Annotators could mark potential misconceptions using pre-defined tags or create new tags, and leave open-ended comments. The tool with final labelings is available at https://wildchat-website.hpcc.eu.org/.

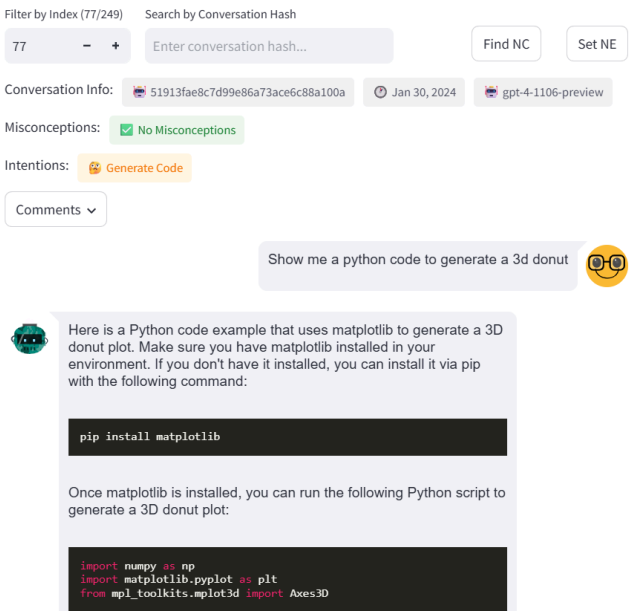

Figure 1: Annotation tool for labeling conversations.

4.4 Codebooks for Conversational Logs

Starting from our brainstormed misconceptions, we aimed to create a codebook applicable to real conversations. However, not all brainstormed misconceptions were relevant to the WildChat chatbot. Some were relevant only to specific architectures, like retrieval augmented generation. We began with a simplified initial codebook applied deductively to conversations. Authors 1 and 2 (A1 and A2) each performed open-coding on 250 unique conversations, totaling 500. Multiple labels could apply per conversation. During open coding, A1 and A2 added codes based on notable observations, plus codes for conversations that were: (i) mostly non-English (likely due to automated language detection errors); (ii) lacking discernible user instructions; and (iii) not Python-related (given regular expression filter limitations). We did not code misconceptions for 85 non-English and 7 non-Python conversations. After individually labeling 250 conversations each, A1 and A2 reviewed each other's flagged misconceptions (78 total) and discussed disagreements. Since potential misconceptions were a minority class, we considered this more informative than reviewing the entire dataset together. We performed axial coding to create finalized codes, de-duplicating codes, merging insufficiently distinct ones, and removing unobserved codes from the initial codebook. A6 resolved outstanding disagreements to create a "Gold" dataset of 50 conversations where at least two annotators agreed on potential misconceptions. This dataset is provided as an artifact. Finally, we conducted a validation round with the remaining authors not involved in Gold dataset creation (A3, A4, A5). We created a stratified validation set of 29 conversations ensuring at least one example of each codebook misconception (labeled in paper artifacts). Annotators could choose multiple codes per conversation and knew each had been flagged as a potential misconception by at least two other authors. Outcomes are presented in Results. The axial codebook is provided in Table 1, and labeled conversations are discussed in Results. We provide an artifact mapping conversation indices (C1, C2, etc.) to identifying hashes from WildChat.

4.5 Measuring Inter-Rater Reliability

We used a modified Krippendorff's α to summarize inter-rater reliability. Different from Cohen's κ, Krippendorff's α is flexible regarding multiple annotators (not pairwise) and is known for its ability to adjust itself to small sample sizes [43]. Since conversations could have multiple non-exclusive labels, we considered "agreement" to occur when two annotators assigned at least one overlapping label. For example, if A1 tagged a conversation with "Web access" and "Non-text output" and A2 tagged it as "Web access", this counted as agreement. Krippendorff's α was calculated for all conversations using this definition, providing a (somewhat optimistic) reliability estimate. Reliability statistics are presented in Results, and the 'agreement' assumption is discussed in the Limitations.

5 Results

5.1 Programming-Related Conversations

The user misconception codebook, with counts from the Gold dataset, is shown in Table 1. Inter-rater reliability (see Section 4.5) on misconception labels suggests agreement well above chance, but with evidence that the coding strategy requires continued refinement. After open coding, A1 and A2 reached $\alpha = 0.539$; after axial coding together, we reached $\alpha = 0.801$ on the Gold dataset. In the validation round (A3, A4, A5), individual agreement with validation labels was lower ($0.433 \leq \alpha \leq 0.504$). For reference, $\alpha \geq 0.8$ is generally considered strong evidence of agreement and $0.667 <= \alpha < 0.800$ is considered moderate [24].

After the final round, we reviewed annotation notes to understand disagreements. Common causes included: difficult-to-parse language from non-native English speakers (the minority of WildChat users are from primarily English-speaking countries [51]), and copy-pasted homework instructions, leading to disagreement about treating user-written versus externally-written prompts.

Most substantially, during Gold dataset creation, A1, A2, and A6 flagged any conversation where a misconception seemed plausible — if not certain. During validation, remaining annotators reserved

Table 1: Codebook for user misconceptions in programming-related WildChat conversations.

Misconception Category	Description	Count	Example(s)
Web access	The prompt asks the LLM to access information located at a particular URL, such as data or code, or requires knowledge of the HTML of a website in the context of webscraping.	20	"Write me a webscraper that will pull the top 100 most popular wikipedia page titles and their visit counts and save it in a csv".
Dynamic analysis	The prompt asks for specific code-checking actions that cannot be fulfilled via static text analysis alone (*e.g.,* runtime errors in Python)	14	"Remove all errors from this program."
Algebra	The prompt asks for an algebraic statement to be evaluated, or would require one to fulfill this.	7	"What is the value of <algebraic expression>?"
Code execution	The prompt requests the output of running a program or asks for the program to be executed	4	"Run this program and tell me the output."
Session memory	The prompt refers to information from a separate conversation as if the LLM has access to that conversation.	4	"Can you add <new feature> to the code you suggested in our last conversation?"
Non-text output	The prompt asks the LLM to return something that is not text, such as an image.	3	"Give me a graph of <company> stock value for the last X years."
Local machine access	The prompt asks for other software tools to be run on the user's machine	3	"Can you run this in my terminal?"
Restrict sources	The prompt asks the model to limit the kinds of information or sources used to generate the output.	2	"Only use code from <repository address> to come up with your answer."
Continuous training	The prompt asks the LLM to use the "latest" or "up to date" information.	1	"Write a script using the latest version of <library>."
Clear chat	The prompt asks the model to delete information or clear the chat.	1	"Clear the chat."
Other agentic behavior	The prompt asks the LLM to take an agentic action, such as asking the model to take a screenshot, play a sound, or use software on an unspecified machine.	1	"Using visual studio code ..."

"misconception" labels for clear-cut cases. However, all annotators agreed that the codebook was reasonable. In other words, some conversations had misconceptions confidently identified by multiple annotators, while other likely misconceptions could not be confidently labeled at the individual conversation level.

We present results in two sections: first, key themes from conversations with reasonably high agreement (where at least one annotator not involved in Gold dataset creation agreed with the Gold label); second, common themes from more ambiguous conversations where agreement did not naturally occur.

5.2 High-Agreement Misconceptions

Web access. Overwhelmingly, the most frequently agreed-upon misconception concerned web access capabilities.

- In C1, a user provided a dataset link and requirements for a system to process linked images and detect fractures. The user indicated "just coding" and the chatbot returned a Python program. However, without details about the dataset format or file structure, the chatbot lacked sufficient information to construct a program that could successfully read the dataset and labels.
- In C2, the user provided a .csv filename and GitHub repository link, asking for a line plot of a specific column (also flagged as expecting non-text output).
- In C14, the user instructed the chatbot to "write a code [sic] to interact with <link to a GitHub page> with python."
- In C18, the user prefaced a prompt with "from the data provided here" followed by a GitHub repository link, then asked about controlling Wi-Fi enabled smart devices.
- In C24, a user prompted the chatbot to "use a genetic algorithm to optimize the features and parameters of a machine learning model for predicting the survival of passengers aboard the Titanic," then provided a link to the Kaggle titanic dataset.

For the Titanic dataset prompt (C24), annotators were confident the dataset would be well represented in the models' training data. We considered this a misconception because the user included a URL. An edge case involved users requesting web scraping programs (C28, C29). For example, C28 prompted "web scrape <link to an IMDb list of top 250 movies> with python and beautifulsoup." Beyond simply pulling HTML, the model would need information about the website's HTML to know which tags relate to the movie list. The chatbot returned a script based on presumed class tags, but it is unclear if these would be correct.

Non-text output. The models used for WildChat were text-only for inputs and outputs, so prompts requesting figure generation (without indicating code would be acceptable) were flagged as potential misconceptions. C7 asked, "give me a meteogram in port of genoa on June 7, 2020." Similarly, C2 asked for a "line plot" without specifying they wanted code, though this appeared to be a homework assignment and the user accepted a code response.

Session memory. We observed one clear example of a user referencing a previous session, which WildChat does not support. C4 asked, "Okay thank you please from the previous programming problem you solved please redo it and round the answers in dollars and cents."

Code execution. Two conversations involved prompts requesting the chatbot to execute programs. In C3, after the chatbot provided a Python program, the user followed up with "make the code run and check for input 5." In C26, the user prompted with a homework assignment containing a Python program and instructions "Click Run and watch the stage to see what's wrong." The chatbot explained why the program "is not running," and the user asked

"did you call the function to make it run?" Curiously, in C334 the chatbot printed simulated program output unprompted.

Local machine access. A user debugging a program (C12) began with "please help" and a traceback (without the program itself). The chatbot responded with solutions including checking internet connection and firewall settings. The user asked, "where did it download the file to or attempt to?" Most annotators considered this evidence of expecting local machine access, though an alternative interpretation is the user asking about default HuggingFace download locations as documentation lookup.

Continuous training. The models have knowledge cutoffs, so requests to use the "latest" version of libraries or APIs were considered misconceptions. In C6, a user prompted "Use latest telethon to delete messges in group that are sent by me," and in C13, "Migrate this code to the latest openai api."

Algebra. C22 prompted, "what does 75% expentional [sic] equate to." Interestingly, the chatbot provided Python code to calculate this value and simulated printing the output ("Output: 2.117...").

Clear chat. In C28, midway through a conversation, the user prompted "clear this page." The bot returned instructions to clear Jupyter notebook output, and the user changed topic. Some annotators disagreed whether this represented context-window-management strategy or misunderstanding that a "clear the chat" system command exists.

5.3 Potential But Unconfirmed Misconceptions

During review of conversation logs, we encountered several challenging themes that emerged as discussion topics. Generally, these were misconceptions we agreed were possible but could not be confidently labeled at the individual conversation level without knowing users' intentions. We discuss them here by theme.

Dynamic analysis. A frequent pattern in debugging conversations was users requesting that *all* bugs be removed from a program. For example: "Fix any bugs in this code and return the full fixed code assembled and ready" (C397), "But I getting ERROR, please give me code which will doesn't give me ERROR" (C19), and "Just make sure the function is error proof" (C278).

Taken literally, it is generally not possible to ensure a program will run without errors only by static inspection–for example, a program could throw an error because a user lacks an installed dependency. We wondered whether users expected the model was capable of dynamic analysis, implying program execution ability. However, we did not feel confident labeling these as misconceptions because this could reflect prompt engineering rather than user beliefs.

Relatedly, we observed conversations where users asked if code would *work* or resolve a previously-discussed bug: "will this one work or no?" (C434), "Review my code now and tell me if it ill work now" (C397). In some cases, it may be reasonable to expect useful input about bug repairs from static analysis–for example, if the user provided a traceback indicating a syntactical error. However, many debugging scenarios require dynamic information.

We note that users often include traceback messages in debugging conversations, providing runtime information to the model. This behavior may evidence that users do not expect a tool can reproduce this information itself.

Optimization. Several conversations focused on optimizing programs for speed or memory: "could you please speed up this function as much as possible?" (C157), "give me time estimation for this code...with ryzen 5 5600x...for this batch_size = 1024..." (C258).

While these conversations do not necessarily indicate misconceptions, we wondered what expectations users have about *how* an LLM-based tool would produce responses. In Python, certain libraries for vectorized computations are almost always faster than algebra on base Python types like lists. This information is likely salient in GPT models' training data, as it is a common help-seeking topic in online forums. Users seeking optimization help may be motivated by mental models about an LLM's training data, or alternatively from incorrect assumptions about the tool's memory profiling capabilities.

Validiation. A relatively common prompt theme was seeking validation that code would meet requirements. Some prompts were worded ambiguously, raising questions about expected validation behaviors. For example, C5 wrote, "i have historical data of crypto as csv files [sic] i have following code to train a model on them without merging them, check if code has any problem or anything wrong." C397 prompted "Review this an tell me what's missing," followed by a code block.

Because these prompts are underspecified, we cannot be confident what kind of "problems" or "missing" elements the user hoped the tool would check for. This prompting style could stem from misconceptions about the tool having access to a "ground truth" about programs to grade them, similar to autograders in computer science education.

Git-like diffs. While iterating on program versions through prompting, users often asked for modifications. For longer programs, WildChat models typically provide a code snippet with the modification and instruct the user to update their program. Sometimes, users requested not just the snippet, but the "full code" (C155, C306)–meaning the original program with the modification made in-line.

This request implies trust that the model will reliably reproduce the entire program, making only the intended modification. Because the previous program version is only represented in the model's context window, a deterministic "diff" is not actually guaranteed. This prompting style could reflect users expecting the model explicitly stores code snapshots as version control, or alternatively reflects trust in the tool's reliability based on reasoning about program length and context window size.

Sensitivity to context window. We noted some conversations involved very long prompts incorporating copy-pasted previous conversations or very long programs. For example, C11 prompted with code and apparent conversation logs totaling 6,021 words (45,345 characters). The model `gpt-4-0125-preview` has a context window of 128,000 tokens–the prompt would fit (in English, a common benchmark is 100 tokens to 75 words). However, model performance is highly sensitive to context window use, and using more is not necessarily better. Prompts of this size could be informed choices or may relate to the "monotonicity belief" [18].

6 Discussion

To summarize our work, we began by brainstorming potential misconceptions that users of conversational LLM-assistants may have in programming contexts, identifying a crucial distinction: misconceptions may concern the affordances of *specific tools* with LLM-components, or LLMs as a class of models. Several papers have demonstrated how LLMs pose challenges for mental model formation [33, 46, 50]; we propose that the variance in affordances of tools built *around* LLMs represents an entirely new frontier of complexity for users to navigate.

A user might have reasonable awareness of core LLM properties, like sensitivity to prompt language and context window usage, but have mistaken ideas about the tools and plugins available to a particular chatbot *using* an LLM as its response mechanism. In our analysis of Python-programming conversations from WildChat, these tool-level misconceptions were most readily apparent, allowing us to surface misconceptions and their manifestations in real-world conversations. We noted potential confusions around the availability of web access, non-text outputs, code execution, algebraic computations, cross-session memory, system commands to clear the chat, and continuous model updates.

We did not observe strong evidence for "model-level" misconceptions in the log analysis, like confusions about context windows. However, such conceptual errors may be less likely to manifest in prompt language. While we note prompts that could indicate misconceptions about the context window, user expectations are not as readily clear from very long prompts as when a user instructs the chatbot to use information from a linked GitHub repository.

Considering the misconceptions we observed about the WildChat chatbot's affordances, many user confusions could be avoided with clearer communication about commonly-available features. Importantly, this must be distinct from the model card for the backend models—the OpenAI API supporting WildChat conversations could optionally use features like web search or "tools", but these may or may not be configured in a given chatbot. If certain features are reasonably common, like web search or code execution, a set of icons or a standardized "specs sheet" might clearly indicate them on the chatbot interface. Importantly, it may not be enough to indicate *when a feature is present*—if users tend to over-estimate chatbot capabilities [16, 21, 23, 39], it may be more important to signal the *absence* of commonly assumed features.

Users may attempt to gain this information by running "system commands" through prompts, like asking "What version of GPT are you?" (our observation that users instruct chatbots to "clear the chat" may be another example of inferring certain system commands exist). However, these prompts may not reliably work, as models may "self-report" fabricated information [9]. We put forward that this information should be clearly communicated through a medium besides LLM-mediated chat interactions.

7 Limitations and Next Steps

The most important limitation of our work is modest inter-rater reliability during annotation, strongly suggesting our codebook requires refinement. Major sources of disagreement included ambiguity from potentially low English proficiency and disagreement about whether certain misconceptions could be confidently recovered from prompt language alone. For example, if a user prompted the chatbot to remove all errors from a program, this could reflect either misplaced assumptions about dynamic analysis capabilities *or* a deliberate prompting strategy. These findings point to necessary revisions before confidently addressing questions like: What are the most common user misconceptions about conversational programming assistants? How do they manifest in prompt language for real-time or post-hoc identification?

To improve measurement validity, we plan to restrict the codebook to "high-agreement" misconceptions from Section 5.2, which are straightforward to observe (e.g., users instructing the chatbot to access URLs). Second, we will test whether labeling becomes easier when restricting conversations to IP addresses from countries with high English proficiency (using metrics like the EF English Proficiency Index [11]). This would reduce global representativeness but may be important for creating an initial expandable contribution.

With a focused codebook, we plan to take a more thorough approach to identifying examples and strengthening the empirical evidence. For high-agreement misconceptions, we will use keyword or fuzzy matching (e.g., via an LLM) to filter conversations with suggestive prompt language–such as URLs or algebraic equations–then annotate only these filtered subsets. This will likely provide more information per conversation than our current study, where non-misconception conversations were the dominant class.

Focusing on confidently-identifiable categories also enables addressing a critical question: what happens *after* a misconceived prompt? How often does the LLM-based tool correct users (perhaps by responding that it cannot access a URL)? If corrective feedback is common, misconceptions may be quickly mitigated. However, if the model appears to comply with impossible requests, misconceptions may persist or be reinforced. We intend to label how frequently WildChat corrects misconceptions within conversations, at least for the most common categories.

Beyond inter-rater reliability, our study has other limitations inherent to the study design. Regarding the WildChat dataset, despite its creators noting that "Since our chatbot is hosted on Hugging Face Spaces, the majority of users are likely associated with the IT community," its representativeness may still be limited. In addition, inferring user intentions from interaction logs is imperfect, as we lack direct access to user expectations. Furthermore, LLM-based tools continue to evolve in their affordances (web access, code execution, cross-session memory). Nevertheless, we expect persistent variation across assistants and believe that our findings can help to ground how such capabilities can be more clearly communicated.

Overall, this paper provides an early empirical characterization of misconceptions programmers hold about conversational LLM-based assistants, identifying concrete gaps between user expectations and tool affordances in programming interactions. In the extended journal publication, we plan to refine the codebook to strengthen empirical validity and more thoroughly examine how such misconceptions are corrected or reinforced by LLM-based tools. By surfacing these misconceptions and their manifestations in real-world conversations, we aim to offer a timely and actionable conceptual foundation to inform tool design and research in LLM-assisted programming.

References

[1] 2025. 2025 Stack Overflow Developer Survey. https://survey.stackoverflow.co/2025/

[2] Alessandra Araújo, Ingryd Nathália, Urbano Vieira, Jessica Nayara, Fernandes Da Silva, Suely Pereira De Faria, Graciele Lorenzoni Nunes, Adibe Georges Khouri, Álvaro Paulo, Silva Souza, Mariana Cristina De Morais, Alexsander Augusto, and D A Silveira. 2017. Automation bias: exploring causal mechanisms and potential mitigation strategies. *Revista Médica del Instituto Mexicano del Seguro Social* 44, 5 (7 2017), 433–440. https://www.redalyc.org/articulo.oa?id=457745535007

[3] Gagan Bansal, Besmira Nushi, Ece Kamar, Walter S. Lasecki, Daniel S. Weld, and Eric Horvitz. 2019. Beyond Accuracy: The Role of Mental Models in Human-AI Team Performance. *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing* 7 (2019), 2–11. doi:10.1609/HCOMP.V7I1.5285

[4] 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 (4 2023). doi:10.1145/3586030

[5] Joel Becker, Nate Rush, Beth Barnes, and David Rein. [n. d.]. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. ([n. d.]).

[6] Michelle Brachman, Siya Kunde, Sarah Miller, Ana Fucs, Samantha Dempsey, Jamie Jabbour, and Werner Geyer. 2025. Building Appropriate Mental Mod-els: What Users Know and Want to Know about an Agentic AI Chat-bot. 18 (2025). doi:10.1145/3708359.3712071

[7] Michelle Brachman, Qian Pan, Hyo Jin Do, Casey Dugan, Arunima Chaudhary, James M Johnson, Priyanshu Rai, Thomas Gschwind, Jim Laredo, Christoph Miksovic, Paolo Scotton, Kartik Talamadupula, Gegi Thomas, Arunima Chaudhary, Tathagata Chakraborti, and Kartik Tala-madupula. [n. d.]. Follow the Successful Herd: Towards Explanations for Improved Use and Mental Models of Natural Language Systems. ([n. d.]), 20. doi:10.1145/3581641.3584088

[8] 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, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. (7 2021). http://arxiv.org/abs/2107.03374

[9] Neil Chowdhury, Daniel Johnson, Vincent Huang, Jacob Steinhardt, and Sarah Schwettmann. 2025. Investigating truthfulness in a pre-release o3 model | Transluce AI. https://transluce.org/investigating-o3-truthfulness

[10] Hyo Jin Do, Michelle Brachman, Casey Dugan, Qian Pan, Priyanshu Rai, James M. Johnson, and Roshni Thawani. 2024. Evaluating What Others Say: The Effect of Accuracy Assessment in Shaping Mental Models of AI Systems. *Proceedings of the ACM on Human-Computer Interaction* 8, CSCW2 (11 2024), 373. doi:10.1145/3686912

[11] EF Education First. 2025. EF English Proficiency Index. https://www.ef.com/wwen/epi/about-epi/

[12] Ahmed Fawzy, Amjed Tahir, and Kelly Blincoe. 2025. Vibe Coding in Practice: Motivations, Challenges, and a Future Outlook - a Grey Literature Review. 1 (2025). doi:10.1145/nnnnnnn.nnnnnnn

[13] Molly Q. Feldman and Carolyn Jane Anderson. 2024. Non-Expert Programmers in the Generative AI Future. *ACM International Conference Proceeding Series* (6 2024). doi:10.1145/3663384.3663393

[14] Kasra Ferdowsifard, Allen Ordookhanians, Hila Peleg, Sorin Lerner, and Nadia Polikarpova. 2020. Small-step live programming by example. *UIST 2020 - Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology* (10 2020), 614–626. doi:10.1145/3379337.3415869

[15] Katy Ilonka Gero, Zahra Ashktorab, Casey Dugan, Qian Pan, James Johnson, Werner Geyer, Maria Ruiz, Sarah Miller, David R. Millen, Murray Campbell, Sadhana Kumaravel, and Wei Zhang. 2020. Mental Models of AI Agents in a Cooperative Game Setting. *Conference on Human Factors in Computing Systems - Proceedings* (4 2020). doi:10.1145/3313831.3376316

[16] Kate Goddard, Abdul Roudsari, and Jeremy C. Wyatt. 2012. Automation bias: A systematic review of frequency, effect mediators, and mitigators. *Journal of the American Medical Informatics Association* 19, 1 (1 2012), 121–127. doi:10.1136/AMIAJNL-2011-000089/3/M{_}AMIAJNL-2011-000089FIG1.JPEG

[17] John P Smith Iii, Andrea A Disessa, and Jeremy Roschelle. 1994. Misconceptions Reconceived: A Constructivist Analysis of Knowledge in Transition. *The Journal of the Learning Sciences* 3, 2 (1994), 115–163. doi:10.1207/s15327809jls0302{_}1

[18] Dhanya Jayagopal, Justin Lubin, and Sarah E. Chasins. 2022. Exploring the Learnability of Program Synthesizers by Novice Programmers. *UIST 2022 - Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology* (10 2022). doi:10.1145/3526113.3545659

[19] Samia Kabir, David N. Udo-Imeh, Bonan Kou, and Tianyi Zhang. 2024. Is Stack Overflow Obsolete? An Empirical Study of the Characteristics of ChatGPT Answers to Stack Overflow Questions. In *Conference on Human Factors in Computing Systems*. ACM. doi:10.1145/3613904.3642596

[20] Andrej Karpathy. 2025. There's a new kind of coding I call "vibe coding".

[21] Majeed Kazemitabaar, Runlong Ye, Xiaoning Wang, Austin Z. Henley, Paul Denny, Michelle Craig, and Tovi Grossman. 2024. CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs. *Conference on Human Factors in Computing Systems - Proceedings* (5 2024). doi:10.1145/3613904.3642773

[22] Markelle Kelly, Aakriti Kumar, Padhraic Smyth, and Mark Steyvers. 2023. Capturing Humans' Mental Models of AI: An Item Response Theory Approach. *ACM International Conference Proceeding Series* (6 2023), 1723–1734. doi:10.1145/3593013.3594111/SUPPL{_}FILE/APPENDIX.PDF

[23] Andrew J Ko, Laura Beckwith Unaffiliated, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Mit Csail, Henry Lieberman, Mary Beth Rosson, and Gregg Rothermel. [n. d.]. The State of the Art in End-User Software Engineering. ([n. d.]).

[24] Klaus Krippendorff. 2006. Reliability in Content Analysis: Some Common Misconceptions and Recommendations. *Human Communication Research* 30, 3 (01 2006), 411–433. arXiv:https://academic.oup.com/hcr/article-pdf/30/3/411/22338169/jhumcom0411.pdf doi:10.1111/j.1468-2958.2004.tb00738.x

[25] Todd Kulesza, Simone Stumpf, Margaret Burnett, and Irwin Kwan. 2012. Tell me more? the effects of mental model soundness on personalizing an intelligent agent. *Conference on Human Factors in Computing Systems - Proceedings* (2012), 1–10. doi:10.1145/2207676.2207678

[26] Sam Lau and Philip Guo. 2023. From "Ban It Till We Understand It" to "Resistance is Futile": How University Programming Instructors Plan to Adapt as More Students Use AI Code Generation and Explanation Tools such as ChatGPT and GitHub Copilot. *ICER 2023 - Proceedings of the 2023 ACM Conference on International Computing Education Research V.1* 16 (8 2023), 106–121. doi:10.1145/3568813.3600138

[27] H. P. H. Lee, A. Sarkar, L. Tankelevitch, I. Drosos, S. Rintel, R. Banks, and N. Wilson. 2025. The Impact of Generative AI on Critical Thinking: Self-Reported Reductions in Cognitive Effort and Confidence Effects From a Survey of Knowledge Workers.

[28] Teemu Lehtinen, Charles Koutcheme, and Arto Hellas. 2024. Let's Ask AI About Their Programs: Exploring ChatGPT's Answers To Program Comprehension Questions. *Proceedings - International Conference on Software Engineering* (5 2024), 221–232. doi:10.1145/3639474.3640058

[29] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. *Proceedings - International Conference on Software Engineering* (2 2024). doi:10.1145/3597503.3608128

[30] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics* 12 (2024), 157–173. doi:10.1162/TACL{_}A{_}00638

[31] Keith McCandless, Arvind Singhal, and Steven H. Cady. 2024. Liberating structures. In *Elgar Encyclopedia of Interdisciplinarity and Transdisciplinarity*. doi:10.4337/9781035317967.ch70

[32] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. (10 2024). https://arxiv.org/abs/2410.05229v1

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

[34] D Norman. 1988. The Psychology of Everyday Things. In *The Psychology of Everyday Things*. Basic Books.

[35] Donald A. Norman. 1999. Affordance, conventions, and design. *Interactions* 6, 3 (5 1999), 38–43.

[36] Gabrielle O'Brien. 2025. How Scientists Use Large Language Models to Program. *Conference on Human Factors in Computing Systems - Proceedings* (4 2025), 16. doi:10.1145/3706598.3713668/SUPPL{_}FILE/PN1358-TALK-VIDEO-CAPTION.VTT

[37] Eduardo Oliveira, Hieke Keuning, and Johan Jeuring. 2023. Student Code Refactoring Misconceptions. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE* 1 (6 2023), 19–25. doi:10.1145/3587102.3588840

[38] Veronica Pimenova, Sarah Fakhoury, Christian Bird, Margaret-Anne Storey, and Madeline Endres. 2025. Good Vibrations? A Qualitative Study of Co-Creation, Communication, Flow, and Trust in Vibe Coding. 1 (9 2025). https://arxiv.org/abs/2509.12491v1

[39] James Prather, Brent N Reeves, Paul Denny, Brett A Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, Gar-Rett Powell, Eddie Antonio Santos, ; P Denny, J Leinonen, A Luxton-Reilly, J Finnie-Ansley, B A Becker, and E Antonio Santos. 2023. "It's Weird That it Knows What I

Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Transactions on Computer-Human Interaction* 31, 1 (11 2023). doi:10.1145/3617367

[40] James Prather, Brent N Reeves, Juho Leinonen, Stephen MacNeil, Arisoa S Randrianasolo, Brett A. Becker, Bailey Kimmel, Jared Wright, and Ben Briggs. 2024. The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. *Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1* (8 2024), 469–486. doi:10.1145/3632620.3671116

[41] Yizhou Qian, James Lehman, Y Qian, and J Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming. *ACM Transactions on Computing Education (TOCE)* 18, 1 (10 2017). doi:10.1145/3077618

[42] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. *ICER 2022 - Proceedings of the 2022 ACM Conference on International Computing Education Research* 1 (8 2022), 27–43. doi:10.1145/3501385.3543957

[43] Gian Luca Scoccia and Marco Autili. 2020. Web Frameworks for Desktop Apps: an Exploratory Study. In *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (Bari, Italy) *(ESEM '20)*. Association for Computing Machinery, New York, NY, USA, Article 35, 6 pages. doi:10.1145/3382494.3422171

[44] Juha Sorva. 2013. Notional machines and introductory programming education. *ACM Transactions on Computing Education* 13, 2 (2013). doi:10.1145/2483710.2483713

[45] Viktoria Stray, Elias Goldmann Brandtzæg, Viggo Tellefsen Wivestad, Astri Barbala, and Nils Brede Moe. 2025. Developer Productivity With and Without GitHub Copilot: A Longitudinal Mixed-Methods Case Study. (9 2025). https://arxiv.org/abs/2509.20353v1

[46] Lev Tankelevitch, Viktor Kewenig, Auste Simkute, Ava Elizabeth Scott, Advait Sarkar, Abigail Sellen, and Sean Rintel. 2024. The Metacognitive Demands and Opportunities of Generative AI. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery. doi:10.1145/3613904.3642902

[47] Christoph Treude and Marco A. Gerosa. 2025. How Developers Interact with AI: A Taxonomy of Human-AI Collaboration in Software Engineering. (1 2025). https://arxiv.org/abs/2501.08774v2

[48] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. *Conference on Human Factors in Computing Systems - Proceedings* (4 2022). doi:10.1145/3491101.3519665

[49] Thomas Weber, Maximilian Brandmaier, Albrecht Schmidt, and Sven Mayer. 2024. Significant Productivity Gains through Programming with Large Language Models. *Proceedings of the ACM on Human-Computer Interaction* 8, EICS (6 2024). doi:10.1145/3661145

[50] J. D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. *Conference on Human Factors in Computing Systems - Proceedings* (4 2023). doi:10.1145/3544548.3581388

[51] Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. 2024. WildChat: 1M ChatGPT Interaction Logs in the Wild. In *The Twelfth International Conference on Learning Representations*. https://arxiv.org/abs/2405.01470v1

[52] Yangtian Zi, Luisa Li, Arjun Guha, Carolyn Anderson, and Molly Q Feldman. 2025. "I Would Have Written My Code Differently': Beginners Struggle to Understand LLM-Generated Code. (6 2025), 1479–1488. doi:10.1145/3696630.3731663

[53] Albert Ziegler, Eirini Kalliamvakou, X Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. Association for Computing Machinery, 21–29. doi:10.1145/3520312.3534864