

“My productivity is boosted, but ...” Demystifying Users’ Perception on AI Coding Assistants

Yunbo Lyu[†], Zhou Yang^{‡§*}, Jieke Shi[†], Jianming Chang[§], Yue Liu[†], David Lo[†]

[†]Singapore Management University [‡]University of Alberta [§]Alberta Machine Intelligence Institute [¶]Southeast University
Email: {yunbolyu, jiekeshi, liuyue, davidlo}@smu.edu.sg, zhou.yang@ualberta.ca, jianmingchang@seu.edu.cn

Abstract—This paper aims to explore fundamental questions in the era when AI coding assistants like GitHub Copilot are widely adopted: *what do developers truly value and criticize in AI coding assistants, and what does this reveal about their needs and expectations in real-world software development?* Unlike previous studies that conduct observational research in controlled and simulated environments, we analyze extensive, first-hand user reviews of AI coding assistants, which capture developers’ authentic perspectives and experiences drawn directly from their actual day-to-day work contexts. We identify 1,085 AI coding assistants from the Visual Studio Code Marketplace. Although they only account for 1.64% of all extensions, we observe a surge in these assistants: over 90% of them are released within the past two years. We then manually analyze the user reviews sampled from 32 AI coding assistants that have sufficient installations and reviews to construct a comprehensive taxonomy of user concerns and feedback about these assistants. We manually annotate each review’s attitude when mentioning certain aspects of coding assistants, yielding nuanced insights into user satisfaction and dissatisfaction regarding specific features, concerns, and overall tool performance. Built on top of the findings—including how users demand not just intelligent suggestions but also context-aware, customizable, and resource-efficient interactions—we propose five practical implications and suggestions to guide the enhancement of AI coding assistants that satisfy user needs.

Index Terms—AI Assistant, IDE, User Experience, Programming, Human-Computer Interaction, Human Factors

I. INTRODUCTION

AI coding assistants,¹ such as GitHub Copilot, have revolutionized software development practices [1]. These assistants leverage Large Language Models (LLMs) to provide code suggestions, ranging from token-level to block-level code completion, bug fixing, testing, and code explanation—covering nearly every aspect of software development [2]–[4]. Their ability to accelerate software development and reduce workloads has led to widespread adoption [5], [6].

Although AI coding assistants have seen rapid growth and widespread adoption, recent studies indicate that developers may spend over 50% of their coding time verifying AI-generated suggestions, which can increase cognitive load and introduce additional validation effort [1]. This highlights the importance of understanding software developers’ perceptions and needs. Such understanding enables assistant developers to better align their tools with actual user requirements and shape the future of software development more effectively.

* Corresponding author.

¹In this paper, the terms AI coding assistants, in-IDE AI coding assistants, assistant, and AI extensions are used interchangeably, referring to AI-powered extensions in the IDE, such as GitHub Copilot.

Recent studies explore developers’ perceptions through interviews [7]–[9], surveys [5], [10], and user studies [1], [11], [12]. However, these studies face limitations, such as small or restricted samples, high costs [13], predefined questions, or a narrow focus on only a few AI coding assistants [14].

Meanwhile, online resources such as app reviews have proven effective in bridging the gap between developers and users, as they often involve large user bases and extensive feedback [15]–[17]. Yet, few studies have mined reviews for AI coding assistants [18], [19], and those that do face three notable limitations: (1) They focus exclusively on GitHub Copilot, overlooking the broader ecosystem of AI coding assistants and missing opportunities for cross-tool comparisons. (2) Existing taxonomy are built from sources like GitHub Issues and Stack Overflow, which highlight technical problems raised by advanced users [20]. This limits insight into subjective experiences and positive feedback, failing to capture what users value—not just what they find problematic. (3) Novice or non-technical users, who may lack the skills or motivation to create GitHub issues, are more likely to leave reviews on the Visual Studio Code Marketplace, where feedback is easier to submit. These overlooked voices are essential for understanding real-world perceptions of AI coding assistants [20], [21].

An unprecedented opportunity exists to address this research gap: thousands of AI coding assistants are available in the Visual Studio Code (VS Code) marketplace, accompanied by extensive user feedback. These real-world reviews provide a rich source of information from developers all over the world, offering insights into their needs and expectations. As the most widely used IDE globally—adopted by 74% of developers according to Stack Overflow’s 2024 survey [22]—VS Code hosts a diverse range of AI coding assistants [23]. This rich dataset enables comprehensive analysis of user perceptions across multiple assistants, offering insights at scale and providing robust validation and extension of prior findings.

We first collect 1,085 AI coding assistants identified among 66,053 extensions in the VS Code marketplace [24] through a combination of manual and automatic labeling. We employ GPT-4o to assist in the labeling process and validate its performance on a statistically significant sample, achieving both precision and recall above 96%. Our analysis reveals that over 90% of AI extensions were released in 2023 and 2024, highlighting a recent surge in their adoption. Additionally, we find that a small number of AI extensions dominate the market, with the top 10 installed extensions accounting for 86% of total installs. Moreover, we find that AI coding assistants receive

more engagement, as AI extensions receive significantly more feedback and have higher average installs.

The first goal of this paper is to construct a taxonomy of the aspects users discuss regarding AI coding assistants. Such a taxonomy helps developers understand what users care about at a high level and which specific characteristics of AI extensions they are particularly concerned with. It also supports improvements in development, maintenance, and testing by highlighting key focus areas. To ensure a systematic taxonomy construction process, we follow best practices for app review analysis [25]. We begin by sampling 361 user comments—statistically representative of the 5,908 reviews collected from 32 AI extensions. We do not consider extensions with insufficient installations or reviews. During the coding process, we adopt a bottom-up merging approach following prior work [26]. We first assign one or more labels to each comment, then merge identical labels, group them into subcategories, and finally organize them into top-level categories. Instead of open card sorting, we use a hybrid card sorting [27], as the top-level categories are informed by established literature in the mature field of app review mining. In our pilot coding of 30 reviews, we initially generated 11 fragmented top-level categories. The predefined structure helps ensure conciseness, with pilot labels naturally aligning with the chosen dimensions. The final taxonomy includes 8 top-level categories (functionality, usability, dependability, supportability, system performance, general experience, comparison, and pricing), 16 subcategories, and 62 distinct labels.

We analyze user feedback on AI coding assistants and identify six key findings. Users value accurate suggestions but often criticize redundant, incomplete, or buggy outputs. While many report productivity gains, novices are positive than experienced developers. Context awareness is a major issue: assistants understand code well but often fail to retain or fetch relevant context. Usability also matters—complex onboarding and disruptive interfaces frustrate users. Response time is generally acceptable, but resource consumption is a common complaint. Finally, pricing and ethical concerns affect adoption, with users favoring free tools and questioning monetization of open-source data.

The contributions of this paper are: (1) Identifying 1,085 AI-related extensions from 66,053 VS Code extensions using a hybrid labeling method. (2) Constructing a three-level taxonomy from 361 user reviews, with 8 categories, 16 subcategories, and 62 labels. (3) Performing sentiment analysis to uncover six key findings on user preferences and concerns, leading to five practical implications. (4) Releasing all data, annotation tools, and results in the replication package.

II. RELATED WORK

A. Studies on User Perceptions of AI Coding Assistants

AI coding assistants are IDE extensions that leverage large language models (LLMs) to support a range of development tasks, including code completion, bug fixing, testing, and code explanation [2]. These assistants primarily operate through three interaction modes: code completion (ranging from token-

to statement-level), chat interfaces, and the emerging agent mode, which can execute terminal commands [28].

Researchers have employed diverse methods to study developers’ perceptions of AI coding assistants, including interviews, surveys, user studies, and mining online content [29]. Zhou et al. [19] mine 476 GitHub issues, 706 discussions, and 142 Stack Overflow posts to build a taxonomy of problems developers encounter with GitHub Copilot. Zhang et al. [18] analyze similar sources to identify developer practices, challenges, and expectations.

Survey and interview studies are common for capturing user experiences. GitHub’s industry report [30] surveys 500 developers on the impact of AI assistants. Liang et al. [5] and Sergeyuk et al. [2] survey 410 and 481 developers respectively to examine usage patterns, avoidance reasons, and desired improvements. Wang et al. [10] combine interviews (15 professionals) and a survey (599 developers) to assess expectations around code completion. McNutt et al. [7] interview 15 data scientists on AI assistant design for notebooks, while Ziegler et al. [31] link Copilot usage metrics to self-reported productivity.

User studies provide insights into interactions. Vaithilingam et al. [11] study 24 participants’ usage of GitHub Copilot. Barke et al. [12] observe 20 developers and identify “acceleration” and “exploration” modes. Mozannar et al. [1] track 21 participants and identify 12 Copilot-related activities. Shah et al. [32] study student use of Copilot’s chat in large codebases and propose the “one-shot prompting” interaction pattern.

Our study analyzes user reviews from the VS Code Marketplace to construct a comprehensive taxonomy of topics users discuss. Compared to the existing taxonomy by Zhou et al. [19], which focuses largely on dependability issues, our taxonomy not only encompasses their identified concerns but also offers broader coverage. Specifically, we uncover user-valued aspects and problems that their taxonomy overlooks. Notably, three of our top-level categories—general experience, pricing, and comparison—capture user sentiments that are absent from their framework. In addition, our findings verify certain contradictions in the current literature and challenge some existing conclusions, as discussed in Section V.

B. Mining App Reviews

App stores have become vital platforms for software distribution, shaping engineering practices by connecting developers and users [17], [25]. While prior research has extensively analyzed app reviews through information extraction [33], classification [34], sentiment analysis [35], and clustering [15]. Classification and clustering are particularly relevant to our study. Classification assigns reviews to predefined categories—such as user intentions [36] or functional versus non-functional requirements [37]—while clustering uncovers themes or shared app characteristics without predefined labels [15], [35]. Both manual [15] and automated [34] approaches have been widely adopted.

III. COLLECTING AI CODING ASSISTANTS

A. VS Code AI Extension Collection

We begin by defining what constitutes an AI extension in the VS Code Extension Marketplace [24]. An extension is considered an AI extension if it includes at least one AI component that assists its functionality. For instance, the GitHub Copilot Extension [38] is considered an AI extension, as it leverages the AI model to provide intelligent code completions or suggestions while developers code. In contrast, we do not classify Azure Machine Learning as an AI extension, as it serves primarily as a tool for developing AI—supporting the building, training, and deployment of machine learning models—rather than incorporating AI into its functionality.

As shown in the top-left of Fig. 1, we follow a two-step process to collect AI extensions from the VS Code Marketplace: (1) collecting AI extensions and (2) hybrid labeling. We collect all potential AI extensions during the first step, and then we confirm the AI extensions in the second step.

AI Extension Collection. We collect metadata for 66,053 extensions available on the Microsoft VS Code Marketplace [24] as of November 28, 2024. To identify AI-related extensions, we examine several sources of information, including metadata (tags and categories) and extension descriptions. For example, the GitHub Copilot Extension [38] includes tags such as “ai” and “openai,” and belongs to the AI category. Its description further states: “*GitHub Copilot provides autocomplete-style suggestions from an AI pair programmer as you code,*” which strongly suggests the presence of AI capabilities. In addition to these sources, the VS Code Marketplace provides an official search feature [39], which returns extensions related to the keyword “AI.”

To ensure comprehensive coverage, we adopt a multi-faceted approach to collect potential AI extensions using the following four methods: (1) *Official Search.* We use the official Marketplace search feature with the keyword “AI”, resulting in 1,623 candidate extensions. (2) *Category-Based Selection.* Each extension in the Marketplace includes predefined categories in its metadata. There are 20 categories in total. Two authors review and select the categories ‘AI’, ‘Chat’, and ‘Machine Learning’ as indicative of AI functionality. This approach yields 740 additional candidate extensions. (3) *Tag-Based Selection.* Tags offer finer-grained metadata about extensions. We find a total of 49,733 unique tags across all extensions. Focusing on the tags that appeared more than 100 times, two authors select nine tags likely to indicate AI content: ai, gpt, chatgpt, openai, copilot, autocomplete, intellisense, deep learning, and machine learning. Extensions containing any of these tags contribute 1,052 entries to our candidate pool. (4) *Descriptions Keyword Matching.* We download the descriptions of all extensions and apply keyword matching using a list of 55 AI-related terms, adapted and refined from prior work [23], [40] The full list is available in the replication package. Matching these keywords against extension descriptions results in 715 candidates.

After aggregating results from the four collection methods and removing duplicates, we identified a final set of 1,962

potential AI extensions. The individual contributions were as follows: 75 from the official search, 345 from categories, 559 from tags, and 414 from descriptions. Searching only through the official search may not identify all AI extensions.

Hybrid Labeling. We first randomly sampled 322 extensions from the pool of candidate AI coding assistants. The sample size is calculated using a sample-size calculator [41], based on a 95% confidence level and a 5% margin of error, a setting commonly adopted in previous work [42]–[44]. Two authors independently label the sampled extensions as either confirmed AI extensions or not, based on predefined criteria. Each extension’s description is reviewed, and documentation is consulted when necessary. The labeling achieves a Cohen’s Kappa coefficient of 0.92 [45], indicating a high level of agreement [46]. Discrepancies are resolved through discussion. In total, among the 322 sampled extensions, 195 are confirmed as AI extensions, while 127 are not.

To scale the labeling process, we explore the capabilities of GPT-4o. We input the extension name and description into GPT-4o and ask it to determine whether the extension qualifies as an AI extension. We experiment with various prompting strategies. On the manually labeled set, we find that a zero-shot learning approach provides good performance, achieving a precision of 96.37% and recall of 96.88%. Manual inspection of misclassifications shows that most errors stem from vague or insufficient descriptions; none involve highly installed or highly rated extensions. We put the prompt used in this study in the replication package due to the limited space. Using the final prompt and setting the temperature to 0 for deterministic output, we label the remaining potential extensions with GPT-4o. This results in 1,085 confirmed AI extensions—55.3% of the initial candidate set—representing 1.64% of all extensions on the VS Code Marketplace.

B. Dataset Overview

Despite the small share of AI extensions (only 1.64% of all extensions), we notice that AI extensions receive more attention, with an average of 124K installs, double that of non-AI extensions (57K). Additionally, AI extensions receive significantly more feedback, with an average of 7.48 ratings per extension compared to 1.76 for non-AI extensions ($p < 0.05$, Mann-Whitney U test [47]). Both AI and non-AI extensions show similar inequality in installs and ratings, as indicated by high Gini coefficients [48] (0.986 vs. 0.982 for installs; 0.931 vs. 0.873 for ratings). The Gini coefficient quantifies statistical dispersion, with 0 indicating perfect equality and 1 representing maximal inequality. However, attention with AI extensions is even more concentrated: the top 10 most-installed AI extensions account for 86% of total installs, and the top 30 most-rated receive 75% of all user ratings. This indicates that a small number of AI extensions dominate the market.

AI extensions have seen rapid growth in recent years, with 90.2% released between 2022 and 2024. Nearly half appeared in 2023 (44.6%) and 2024 (45.6%), highlighting their recent surge. In contrast, non-AI extensions have followed a more gradual distribution, with moderate growth in recent years

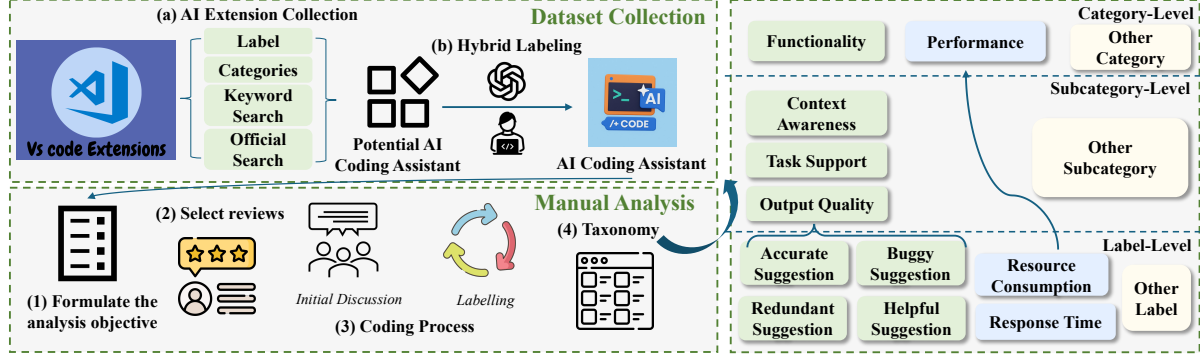


Fig. 1: Our five-step process for analyzing reviews from the VS Code marketplace: (1) formulate analysis objective, (2) select reviews for analysis, (3) specify unit of analysis, (4) perform coding process, and (5) analyze dataset.

(17.8% in 2023, 18.2% in 2024) and a significant portion released in earlier years. This contrast underscores the sudden rise of AI-powered extensions, whereas non-AI extensions have evolved at a steadier pace over time.

IV. STUDY DESIGN

A. Research Questions

We aim to understand user perceptions of AI coding assistants by address the following research questions:

- **RQ1. What do users discuss about AI coding assistants?**
- **RQ2. What aspects of AI coding assistants do users care about, appreciate, and dislike?**

RQ1 seeks to identify and categorize the main topics users discuss about AI coding assistants. RQ2 focuses on understanding user preferences, including what aspects of AI coding assistants they care about, appreciate, or dislike. To build the taxonomy and systematically understand user preferences, we follow the process suggested by Dąbrowski et al. [25] for analyzing app reviews in software engineering. The process consists of the following steps, as illustrated in the lower-left of Fig.1: (1) formulate the analysis objective; (2) select reviews for analysis; (3) specify the unit of analysis; (4) perform the coding process; and (5) analyze the dataset.

B. Research Object

Formulate the analysis objective. To ensure the extraction of meaningful insights, we focus on a strategically selected subset of AI extensions rather than the full set collected in Section III. Specifically, we target two types of AI extensions with broad user reach and engagement: (1) those with the highest number of installs, and (2) those with the highest number of user reviews. For the first criterion, we select the 30 most-installed AI extensions, which collectively account for 95.3% of total installs and 68.5% of all user reviews. For the second, we include all AI extensions with more than 100 user reviews, as they demonstrate high user engagement. This adds two additional extensions not included in the top 30 by install count. In total, we analyze 32 AI extensions, balancing

popularity (install count) and user engagement (number of reviews) to support a comprehensive qualitative study.

Select reviews for analysis. We analyze user feedback to identify the topics users discuss regarding AI coding assistants. From a total of 5,908 reviews across 32 AI extensions, we exclude 649 reviews that contain fewer than 10 words, as such short reviews are unlikely to provide meaningful insights. This results in 5,259 reviews. Given the large volume, analyzing all 5,259 reviews is impractical. Therefore, we randomly sample 361 reviews, ensuring a 95% confidence level with a 5% margin of error. This approach is consistent with prior studies [42], [44]. Feedback in all languages is included to capture diverse perspectives (as we discussed in Section V, users provide region- or country-specific comments), and translate non-English reviews into English using Google Translate.

Specify unit of analysis. We use the individual review as the unit of analysis. Each review serves as a standalone piece of feedback that conveys a complete user experience, allowing for consistent manual interpretation and coding.

C. Coding Process

We structure the coding process in two stages: an *initial discussion* to establish coding principles, followed by *iterative coding* to develop and refine the taxonomy.

Initial discussion. Before initiating substantive coding, four participants—three PhD candidates and one senior researcher with expertise in SE and AI—jointly reviewed a sample of 30 user reviews drawn from the full set of 361. Through discussion, we establish four guiding principles: (1) *Multi-label annotation*: A single review often addresses multiple independent themes—for instance, praising an assistant’s helpful suggestions while criticizing its resource consumption. Therefore, we allow multiple labels per review when separate topics are discussed. (2) *Three-level taxonomy*: A two-level hierarchy (category–leaf node), as used in prior work [19], proves insufficient for capturing the content observed in our pilot sample—too many leaf nodes fall under a single category, resulting in overly broad themes. We therefore adopt a three-

level taxonomy (category–subcategory–leaf node) to better represent topic granularity, in line with prior research [26].

(3) *Bottom-up merging process*: We adopt a bottom-up method to construct the taxonomy similar to prior work [26]. In each round, two annotators independently assign one or more labels to each review. They then meet to reconcile any disagreements and consolidate semantically similar labels into unified expressions, which we refer to as leaf nodes. These leaf nodes are initially grouped directly under top-level categories, forming a category–leaf node structure. A subcategory layer is introduced only when a category contains a sufficiently diverse set of leaf nodes—typically more than five—resulting in a three-level structure (category–subcategory–leaf node).

(4) *Hybrid card sorting*: We adopt a hybrid card-sorting approach to provide an initial structure while retaining flexibility for refinement. Specifically, we predefine five top-level categories—*Functionality*, *Usability*, *Dependability*, *Supportability*, and *Performance*—based on established taxonomies from the app review literature [37], [49]. This choice is motivated by two considerations. First, app review mining is a mature field, and its taxonomies are derived from large-scale datasets covering diverse software types, offering generalizable top-level categories. Second, when reviewing our pilot sample, we found that most user comments naturally fit within these predefined dimensions. Without such a structure, our initial coding produced 11 top-level categories from just 30 reviews, resulting in a fragmented and inconsistent taxonomy. All remaining subcategories and leaf nodes are derived through bottom-up consolidation, with annotators free to revise or expand the taxonomy as needed throughout the coding process.

Iterative coding. We conduct annotation in iterative batches of 50 user reviews. In each round, two annotators independently label reviews based on principles set in the initial discussion. After three rounds, we identify a gap in the taxonomy: while technical and functional aspects are well covered, broader sentiments—such as perceptions of productivity or satisfaction—are underrepresented. To address this, we revisit earlier reviews to ensure no content related to users’ general impressions is overlooked, resulting in three new categories: *general experience* (e.g., productivity boost), *pricing* (e.g., free to use), and *comparison* (e.g., evaluation against other tools). By the end of Round 3, the taxonomy includes 8 categories, 16 subcategories, and 55 leaf nodes.

In Rounds 4 through 6, the taxonomy stabilizes. Only five additional leaf nodes emerge across the three rounds (1, 2, and 2, respectively), and no new categories or subcategories are introduced—indicating structural saturation [26]. At this stage, we finalize the taxonomy and compile a formal annotation guideline, including definitions and illustrative examples for each category, subcategory, and leaf node.

The two annotators independently label the remaining 17% of the dataset. Disagreements are resolved through discussion. The inter-rater agreement for this final phase is $K = 0.983$, indicating strong consistency between annotators [46]. Only two additional labels are introduced in this phase. In total, we annotated 361 user reviews, resulting in 62 unique leaf nodes,

structured across 16 subcategories and 8 top-level categories. Note: In the later content, “label” refers to the leaf nodes.

D. Sentiment Analysis

A single review may express multiple sentiments—for example, a user may praise helpful suggestions but criticize high memory consumption. Therefore, instead of assigning one sentiment per review, we analyze sentiment at the leaf node level. Although leaf nodes are finalized through bottom-up consolidation, sentiment is annotated when individual labels are first applied. Each label is marked as like, dislike, or neutral (including feature requests or emotionally neutral mentions). In the first six rounds, two annotators jointly labeled sentiment during coding. In the final phase (17% of reviews), sentiment was annotated independently. We report inter-rater agreement only for this phase, as earlier rounds involved evolving labels and collaborative decisions. Agreement for the final phase is $K = 0.943$, indicating high consistency [46].

V. CATEGORIZE DISCUSSION OF CODING ASSISTANTS

In this section, we present the taxonomy derived from our analysis of user reviews of AI coding assistants in VS Code marketplace. The taxonomy results are presented in Table I. We categorize topics of user reviews into eight high-level categories: *functionality*, *usability*, *dependability*, *system performance*, *supportability*, *general experience*, *pricing*, and *comparison*. In total, we identify 8 categories, 16 subcategories, and 62 labels. In the remainder of this subsection, we define each category along with its corresponding subcategories and labels. We report the frequencies leaf nodes using the multiplication symbol (\times).

A. Functionality

The functionality category refers to user discussions about the core features of the AI coding assistant, including five subcategories: Suggestion Content, Language, Library, Task Support, Context Awareness, and Understandability, and Development Environment Integration. It has a total of 18 labels.

1) *Suggestion Content*: This subcategory covers a set of users’ opinions on suggestions from the coding assistants. The most frequently discussed label is *accuracy of suggestions* ($43\times$), reflecting how suggestions match the user’s expectations. The second most discussed label is *helpfulness of suggestion* ($32\times$), indicating whether the suggestion is useful and aligns with the user’s intent—even if not entirely accurate, it may still provide valuable hints. The third most mentioned label is *completeness and redundancy of suggestions* ($11\times$), which concerns whether the output is sufficiently complete or contains unnecessary parts. This is followed by *failure on complex tasks* ($5\times$), referring to the assistant’s limitations when handling more sophisticated programming problems. Next is *buggy suggestion* ($4\times$), which captures instances where the AI produces incorrect or malfunctioning code. Finally, *other* ($3\times$) includes less common issues such as hallucinated content or severely outdated suggestion.

TABLE I: Categorization of discussion of AI Coding Assistants. PL, DEI are abbreviations for Programming Language and Development Environment Integration. The table is sorted by the number of reviews in descending order. The last column represents users’ sentiment toward each category, subcategory, or label.

ID	Category-Subcategory	Description	No.	Rate	Sentiment
1	Functionality • Suggestion Content • PL, Library, Task Support. • Understanding ability • Context Awareness • DEI	Core code-generation and assistance features.	238	32.2%	64% 31%
		Opinions on code suggestion content.	101	13.7%	59% 37%
		Support for languages, libraries, and SE tasks.	73	9.9%	78% 17%
		Ability to understand code and user intent.	45	6.1%	62% 29%
		Ability to leverage code/project context.	21	2.8%	38% 57%
		Integration with tools (terminal, debugger).	7	1.0%	100% 0%
2	General Experience • Productivity • General Discussion • Helpfulness	Overall experience and emotional response.	136	18.4%	90% 10%
		Reported acceleration or slow down of coding speed, flow.	83	11.2%	90% 10%
		Open-ended reflections or pure praise and claims.	28	3.8%	89% 11%
		Usefulness of suggestions solving problems.	23	3.4%	92% 8%
3	Usability • UI & Interactivity • Controllability • Learnability • Predictability	Interface design and ease of use.	104	14.1%	53% 36%
		Interface layout, chat panel, pop-up quality, cursor control.	60	8.1%	55% 30%
		Settings, model choice, interruption control.	21	2.8%	76% 19%
		On-boarding difficulty and docs clarity.	16	2.2%	38% 50%
		Consistency of responses, avoids corruption.	7	1.0%	0% 100%
4	Dependability • Reliability • Legal & Ethical Concerns • Security & Privacy • Availability	Trustworthiness: reliability, security, ethics, uptime.	83	11.2%	19% 77%
		Stability, crashes, install failures, fallbacks.	37	5.0%	13% 84%
		License compliance, AI-ethics concerns.	23	3.1%	22% 74%
		Risk of code leaks, privacy-safeguard adequacy.	12	1.6%	17% 83%
		Offline capability and regional limitations.	10	1.4%	40% 50%
5	Pricing • Free to use • Value Perception	Monetary cost, free tiers, perceived value.	55	7.4%	69% 29%
		Positive reactions to generous free tiers.	32	4.3%	97% 0%
		Overpricing claims versus fair-price praise.	23	3.1%	68% 32%
6	Supportability • Compatibility • Serviceability • Feature Availability • Maintainability	Post-deployment support, compatibility, rollout.	48	6.5%	58% 33%
		OS / IDE / web compatibility, conflicts.	18	2.4%	47% 35%
		Vendor or community support responsiveness.	15	2.0%	93% 7%
		Speed and openness of new-feature access.	11	1.5%	46% 46%
		Stability across updates, maintenance pace.	4	0.5%	20% 80%
7	Comparison • With Competition • With Github Copilot • With GPT	Comparisons with other AI tools.	44	6.0%	71% 27%
		General comparisons to rival products.	21	2.9%	81% 19%
		Contrasts in accuracy, speed, price vs Copilot.	19	2.6%	74% 21%
		Evaluations versus raw ChatGPT / GPT-4.	4	0.5%	0% 100%
8	Performance • Response Time • Resource Consumption • Rate Limiting	Latency, throughput, resource footprint issues.	31	4.2%	58% 42%
		Perceived waiting time between request and output.	17	2.3%	82% 18%
		CPU, memory, battery drain complaints.	9	1.2%	22% 78%
		Complaints about request caps	5	0.7%	40% 60%

Like Request Dislike

2) *PL, Library, Task Support*: This subcategory reflects user discussions about the AI coding assistant’s ability to support specific programming languages (PL), libraries, and tasks. The most discussed label is *task support* (36×), which focuses on capabilities beyond basic code completion—such as test generation, code refactoring, and code summarization. We exclude general code generation since it is a standard feature across all assistants analyzed. The second most discussed label is *PL support* (28×), with users referencing 13 languages. The third discussed label is *library or framework support* (9×).

3) *Understandability*: his subcategory captures users’ concerns about the assistant’s ability to comprehend code, errors, and human intent after retrieving the relevant context. Users frequently highlight the importance of *contextual understanding* (23×), *code understanding* (8×), *human instruction understanding* (6×), *good explanation of the code* (4×), and *error or syntax understanding* (3×), referring to the assistant’s ability to identify issues in the code.

4) *Context Awareness*: his subcategory reflects users’ expectations regarding the assistant’s ability to accurately retrieve and maintain contextual information. *Project/Codebase context awareness* (13×) captures expectations for understanding and navigating across project- or repository-level contexts. *Context memory capacity* (7×) refers to the ability to retain and leverage past interactions.

5) *Development Environment Integration*: Users mention it which refers to the assistant’s ability to work seamlessly with parts of the development environment, including the terminal, version control systems, and debugging tools.

B. Usability

This category includes reviews that discuss the user interface and ease of use of the AI coding assistant. It includes four subcategories: UI & interactivity, learnability, controllability, and predictability, with a total of 12 labels.

1) *UI & Interactivity*: This subcategory includes user feedback on interface design and interaction mechanisms. The

most discussed label is *chat interface* (23×), where users comment on chat-based interaction, which differs from traditional code completion. For example, some users find in-line chat convenient. *General design and user experience* (14×) captures feedback on intuitive, user-friendly interfaces and seamless access to AI functionality. *Suggestion interface & interaction* (13×) includes feedback on how suggestions are presented (e.g., with stars, syntax highlighting, or snooze buttons) and how users interact with them (e.g., using shortcuts or whether suggestions are proactive). *Cursor interactions* (6×) refers to feedback on cursor control, such as assistants changing cursor position unexpectedly or correctly predicting where the cursor should move. *Layout & navigation* (4×) includes comments about layout features like split view, sidebar management, and navigating to functions or class definitions.

2) *Learnability*: This subcategory reflects user discussions about whether the assistant is easy to learn and use. *Onboarding difficulty* (6×) is the most discussed label, referring to challenges in getting started, including the learning curve and setup process. *Ease of use* (5×) reflects user impressions that the assistant is intuitive and includes helpful features such as help bar. *Documentation* (5×) includes comments on unclear or insufficient documentation.

3) *Controllability*: This subcategory reflects user feedback on how much control they have over the behavior and configuration of the AI coding assistant. The most discussed label is *Customization* (12×), where users discuss support for pre-configured and custom commands, configurability settings, and the ability to manage code snippets. *Model choice* (4×) refers to the ability to select different models, including local models or cloud-based ones like GPT-4o. *Extension dependency issue* (4×) includes comments on extensions being automatically installed due to dependencies, sometimes without user consent.

4) *Predictability*: Some users report that the assistant may *mess up their existing code* (6×), introducing unexpected changes. Others mention *inconsistent language output*, such as responses switching between English and French.

C. Dependability

This category includes user reviews that raise concerns about the dependability or trustworthiness of the assistant, covering aspects such as reliability, availability, and security. It contains four subcategories: reliability, legal & ethical concerns, security & privacy, and availability.

1) *Reliability*: This subcategory captures user feedback about the stability and consistent operation of the extension. The most discussed label is *bug of the extension* (11×), where users report crashes, unresponsiveness, or specific bugs. *Fallback to weak model* (6×) includes complaints that the assistant downgrades to a weaker model or becomes less responsive over time, leading to a noticeable drop in answer quality. *Login issue* (6×) refers to problems with logging in or authentication failures. *Installation issue* (3×) includes cases where users are unable to install the extension or find

it unusable after installation. Another issue mentioned is *AI service mistakenly blocking valid queries* (2×).

2) *Legal & Ethical concerns*: *AI ethics* (19×) is the most discussed label, where users raise concerns about open-source culture and AI ethics, such as being charged for models trained on their code. *License compliance* (4×) refers to concerns generated code may violate code licenses.

3) *Security & Privacy*: Users discuss *privacy protection* (6×), such as the risk of data leakage, and *permission over-reach* (3×), where extensions request excessive permissions. One user also mentions concern about potential data being sent to external servers, raising security risks.

4) *Availability*: Users express the need for consistent access to the assistant. Some users complain or request that the extension should *work locally* (5×) without internet connection. Another issue is *region limitation* (5×), where users report that the extension is unavailable in their geographic area.

D. Supportability

This category includes user reviews concerning the assistant's ability to operate across different operating systems and IDEs (i.e., compatibility), as well as concerns related to updates and maintenance (e.g., maintainability). It comprises three subcategories: compatibility, feature availability, and maintainability, with a total of nine labels.

1) *Compatibility*: *IDE compatibility* (17×) is the most discussed label, where users request support for additional IDEs, such as Atom. *Operating system compatibility* (2×) includes comments about extending support to other platforms, such as Linux. Some users also mention that the assistant should be *compatible with the web version* (2×) of the IDE.

2) *Feature Availability*: In this subcategory, users express appreciation for *rapid feature improvements* (11×) but also frustration with the *waiting list for new features* (5×).

3) *Maintainability*: Some users report concerns about the assistant's *lack of maintenance* (1×), the *frequency of updates* (2×), and instances where updates *break functionality* (2×).

Serviceability (15×) is also frequently mentioned, referring to the development team's responsiveness to customer or community support requests.

E. System Performance

This category includes user reviews focused on the system performance of the AI coding assistant, such as response time, resource consumption, and rate limiting. It contains three labels. Users frequently discuss *response time* (17×), expressing appreciation for fast performance while also complaining about occasional unresponsiveness. *Resource consumption* (9×) is another common concern, with users reporting that the assistant consumes excessive system resources, such as high CPU and memory usage. A few users also mention *rate limiting* (3×), referring to restrictions on the number of requests or API calls the assistant can handle.

TABLE II: What Do Users Like and Dislike?

Top-15 Users' Like			Top-15 Users' Dislike		
no.	Label	N.	no.	Label	N.
L1	Accuracy suggestion	39	D1	helpfulness suggestion	17
L2	Task support	24	D2	AI ethics	15
L3	PL Support	20	D3	Bug of the extension	12
L4	Chat interface	16	D4	complet & redundant	8
L5	helpfulness suggestion	14	D5	Resource Consumption	7
L6	Response Time	14	D6	Project Context Support	6
L7	Serviceability	14	D7	Suggestion UI	6
L8	Customization	12	D8	Chat interface	6
L9	Code understanding	11	D9	PL Support	6
L10	IDE Compatibility	8	D10	Context-memory capacity	6
L11	General design	7	D11	On-boarding Difficulty	6
L12	Project Context Support	7	D12	Mess up the code	5
L13	Framework support	6	D13	Fallback to weak model	5
L14	Suggestion UI	6	D14	Frustration waiting list	5
L15	On-boarding Difficulty	5	D15	Login Issue	5

F. Others

1) *General Experience*: This category includes topics not directly related to the functional features of the AI coding assistant, but rather to users' overall experience. It contains one subcategory (productivity) and five labels.

Productivity. As AI coding assistants are designed to enhance user productivity [11], *productivity* (83×) is the most frequently discussed topic. Users often report that the assistant helps *boost their productivity* (73×), while some mention that it can *slow down coding speed* (8×). A few users also state that it helps them *find a solution to their problem* (3×).

In addition, some users engage in more *general discussion* (28×), including reflections on mistake tolerance, concerns about over-reliance on the assistant, or pure *praise or complaints* (25×). Users also comment on the *helpfulness* (25×) of the assistant, noting that it supports developers at various stages of experience.

2) *Pricing*: This category includes user reviews related to the pricing of the assistant. Two labels are discussed. Users often praise the assistant for being *free to use* (33×) and share opinions on its *value perception* (23×), with some considering it overpriced and others finding it worth the cost.

3) *Comparison*: This category includes user reviews that compare the AI coding assistant with other AI tools. It contains three labels. Users compare the assistant with general *competitors* (21×), as well as specifically with *GitHub Copilot* (19×) and *ChatGPT* (4×).

VI. WHAT ASPECTS OF AI CODING ASSISTANTS DO USERS CARE ABOUT, LIKE, AND DISLIKE?

We summarize the insights from user reviews into six findings. We quote original user review texts to support our findings. Each review is referenced using an identifier (e.g., (R1, 5★) refers to the first review in our dataset, where the user gave the AI extension a 5-star rating—the highest possible score in the VS Code Marketplace, which reflects the user's evaluation of the extension itself (not the quality of the review), with 1 being the lowest and 5 the highest.

Finding 1. Users generally perceive that AI coding assistants help boost productivity, though perceptions of

helpfulness vary by experience level. Productivity has been a prominent topic since the introduction of GitHub Copilot [50]. Prior studies report perceived productivity gains among users [10], [11], [51]–[53], though some lab-based experiments find no statistically significant improvements [1], [11]. In our study, 90% of reviews labeled with productivity express positive sentiment, describing how the assistant streamlines workflows, reduces repetitive tasks, and improves efficiency—for example: *“not having to type every single repetitive function out or imports”* (R94, 5★). This aligns with the “acceleration mode” observed in prior work, where developers use the assistant to implement known solutions more quickly [12]. Co-occurrence analysis between labels shows that productivity often appears alongside accurate suggestions (13×), code understanding (7×), PL support (6×), and customization (6×). The first three are expected, as strong contextual understanding and language-specific support contribute to faster development. Customization, on par with PL support, is somewhat unexpected. Upon review, we find that users value the ability to tailor the assistant to their needs, enhancing productivity: *“easily choose the languages you want to focus on this is so valuable”* or *“The ability to customize [assistant]’s suggestions and responses helps developers align the tool with their coding style and preferences...accelerating the coding process.”* Indeed, customization is the 7th most liked label, with 12 mentions (see Table II).

Helpfulness has also been debated in prior research. Some argue that AI assistants benefit experienced developers who understand how to use them effectively [54], while others suggest they are especially helpful to novices [55]. Our findings support the latter: 14 out of 15 junior developers explicitly describe the assistant as helpful—e.g., *“I am a beginner programmer and it is helping me a lot to build a project”* (R319, 5★). In contrast, experienced developers express more skepticism; two out of five report limited value: *“For anyone who really knows how to code, save yourself a lot of frustration”* (R14, 1★).

Finding 2. Users care most about suggestion content. Accurate suggestions are highly valued, but users often criticize redundant, incomplete, or buggy outputs and express mixed views on the helpfulness of suggestions. As shown in Table I, suggestion content is the most frequently discussed topic, accounting for 14% of all labeled mentions. Among these, 59% of reviews express positive sentiment, while 41% are negative. A key aspect users appreciate is the *accuracy of suggestions*, which received 39 positive mentions—making it the most liked label in our dataset (see Table II). One user remarked: *“80% less keyboard touching. Autocomplete is pure magic. Feels like it’s connected directly to your mind”* (R164, 5★). However, users also frequently highlight frustrations, particularly regarding the *helpfulness of suggestions* (17 dislikes), *redundant or incomplete outputs* (8 dislikes), and *buggy suggestions* (4 dislikes), including failures on complex tasks (4 dislikes). Notably, issues with redundancy and incompleteness have received little attention

in prior studies, yet they represent the second most cited reason for negative sentiment within suggestion-related feedback. For example, users report: “*Constantly barfs words on the screen 90+% is repetitive*” (R14, 1★), and “*it only predicts one character for me*” (R34, 1★).

Finding 3. Users concern about context awareness. While AI assistants understand code well given context, they often struggle to retrieve and retain it. Prior studies have highlighted the importance of context understanding in AI-assisted coding [5], and noted that a lack of contextual understanding can deter users from adopting such tools [2]. In our study, however, we find that the core issue is not the assistant’s ability to understand context once it is available, but rather how effectively it fetches or maintains relevant context—i.e., context awareness. As shown in Table I, reviews mentioning contextual understanding—defined as the assistant’s ability to comprehend code and human instructions after the context is retrieved—are mostly positive, with 73% explicitly highlight strong contextual understanding. In contrast, feedback on context awareness—the assistant’s ability to retrieve and maintain relevant context—is more negative, with only 38% of related reviews are positive. Notably, two of the most disliked labels are related to context handling: project/codebase context support and context-memory capacity (see Table II). For example, one user notes: “[assistant] still doesn’t see the class definitions in files that aren’t open” (R1, 1★). Others are frustrated by limited memory, such as: “[assistant] forgets context on next question and answers irrelevantly even for simple questions” (R22, 1★). We also observe that context-memory capacity frequently co-occurs with complaints about unhelpful suggestions (4 times), indicating that limited context retention can directly degrade output quality. For instance, one user write: “What AI forgets between prompts within the same thread. My code looks like Frankenstein given the amount of contradiction because it couldn’t recall what I just recommended previously.” (R79, 1★).

Finding 4. Usability matters to users—minor design issues can cause abandonment. Users expect AI assistants to be easy to learn and use. While previous studies have largely focused on functionality and dependability—such as suggestion quality, privacy, and security concerns [2], [5], [19]—usability issues have received relatively little attention. In our study, however, we find that usability is the second most discussed topic (after functionality), accounting for 12% of all labels. One major usability concern is onboarding difficulty. Although more features and stronger capabilities can enhance productivity, they also increase the learning curve. Among the six complaints related to onboarding, four come from extensions supporting local models, and all involve complex setup processes. For instance: “Setup process is bloated. I’ll wait until they make the process more streamlined.” (R265, 1★). For cloud-based assistants, users report difficulty adapting to AI-driven workflows: “While [assistant] aims to simplify coding, some users might find it challenging to adapt to the AI’s suggestions and functionality, especially if they’re used to

traditional coding practices.” (R240, 4★).

Users also express frustration with suggestion interface and interaction, which are the sixth and twelfth most disliked labels in our dataset. Complaints include unwanted suggestion placement—e.g., “*Annoyed suggestions show up at the top*”—and broken interaction flows, particularly related to cursor focus: “*Focus doesn’t work, making chat useless...frustrated, don’t use this extension.*” (R312, 1★). This issue refers to the assistant’s “focus” feature, which attempts to shift user attention to the chat window. However, users report that this can unpredictably hijack the cursor—causing input intended for the code editor to be redirected to the chat window.² Additionally, some users report severe usability breakdowns where the assistant interferes with their code: “*Messed so much with my code*” (R7, 3★).

Finding 5. Users are dissatisfied with resource consumption but generally satisfied with response time. Previous studies have noted that response time affects user experience, with users preferring statement-level responses within two seconds [10]. In our study, 14 out of 17 reviews express satisfaction with response time, likely due to the increasing use of online models, which typically offer faster responses. Among the 32 extensions we analyzed, only four exclusively support local models. In contrast, users frequently complain about resource consumption. As shown in Table I, 78% of related reviews express dissatisfaction, making it the fourth most disliked label. For example: “*Uses too much resources—over 50% CPU and more than 1 GB memory*” (R125, 1★). While such concerns are often associated with local models, we find that performance complaints also arise with online models, particularly when working on large files or multi-project setups. Among the eight reviews citing high resource usage, only half refer to local models; the rest involve online models. One user notes: “*the extension’s performance can sometimes slow down the editor, especially when working on larger files or multi projects*” (R306, 5★).

Finding 6. Users weigh pricing and feature value when adopting AI coding assistants. Prior studies have paid limited attention to pricing, as their data sources have primarily focused on functional performance [5], [19]. However, in our study, pricing and comparison emerged as a significant concern and plays a clear role in users’ decisions to adopt or abandon AI coding assistants. Many users express a preference for free alternatives over paid assistants. For example: “*It’s a wonderful free alternative of paid AI code assistants*” (R213, 5★). Users are particularly critical when an assistant lacks competitive features yet costs more than alternatives like GitHub Copilot or ChatGPT. One user writes: “*I also tried Github Copilot, and I have to say it’s a hundred levels better than [assistant], and it’s cheaper to subscribe to.*” (R99, 1★). Some users explicitly abandon AI assistants due to pricing concerns: “*Was cool to try out but too expensive now. You are using our code to make money. So, pass for now...but I think you should have a free version (since it’s using*

²<https://www.reddit.com/r/Codeium/comments/1es4pdo>

open source)” (R42, 1★). This also reflects a broader ethical concern: users are dissatisfied with the practice of training on open-source code and then charging for the service. This issue—categorized under AI ethics label—is the second most disliked label in our study.

Other Findings. We summarize additional insights from the remaining content in Tables I and II, and relate them to prior work below. The least liked subcategories—predictability, reliability, and security & privacy—reflect high user dissatisfaction. Reliability and security/privacy concerns are consistent with prior studies, where users reported operational issues and fears of data leakage [5], [19], [56], [57]. In contrast, task support and PL support are the second and third most liked aspects. Users value the assistant’s help with code refactoring and test generation, aligning with Sergeyuk et al. [2], who identified these as top tasks developers would delegate to AI assistants. While PL support is generally praised, some users mention a lack of support for niche languages. The chat interface ranks fourth in user preference. Many users appreciate not having to rely on external searches [58], though some criticize interface layout, history management, and dialog interaction.

VII. DISCUSSION

A. Implication for Improving AI Coding Assistants

Implication 1: Improving context awareness is essential for supporting many tasks. As discussed in Finding 3, users complain about poor context awareness, which leads to unhelpful suggestions. This limitation is especially problematic for tasks like project-level refactoring, which require broader understanding of code dependencies (e.g., function changes, return types, internal logic). For example: “[assistant] needs to support code refactoring...changing function definitions, return types, and function contents to make sure everything compiles is a huge time sink” (R41, 4★). Improving repository-level context support—through codebase indexing or manual linking of relevant files—can help. Some assistants already support this, and users respond positively (e.g., “Indexing your repo code and adding context with @” (R355, 5★)). They can also enhance effectiveness by dynamic inference [59], applying context compression techniques [60], or fine-tuning models based on the user’s repository for more personalized support [61], which can also improve the effectiveness of related tasks like program repair [62], [63].

Implication 2: Chat and agent interfaces need more research attention. While chat and agent-like capabilities are increasingly integrated into coding assistants [28], prior work on understanding how assistants can help programmers has focused mainly on traditional code completion [10], [11]. Few studies examine the usability of chat [32], [58], and none systematically study agent interfaces. There is a need to understand how users interact with these modes and to design more effective interaction patterns. Existing work on suggestion presentation [64], [65] can inform future research.

Implication 3: Usability must be prioritized through intuitive design. Usability was the second most discussed topic in

our study. Users expect intuitive design, clear documentation, and easy onboarding (e.g., “*intuitive design and user-friendly interface*” (R88)). For example, Tabnine clearly communicates its language support levels and uses AI to provide context-aware guidance. As discussed earlier, improving chat and agent interactions is also crucial. Developers should also avoid intrusive features like ads during inference, which significantly harm user experience (R99, 1★). Finally, efficiency is another important aspect valued by users. Assistants with low resource consumption can be achieved through techniques such as model compression [66]–[68].

Implication 4: Developers must ensure dependability to build trust. Perceived dependability strongly influences satisfaction [69]. Users are deterred by instability, bugs, fallback to weak models, and privacy concerns [70]. To build trust, assistants should be stable [71], reduce vulnerabilities [72], ensure fairness [73], and remain transparent. Users also request source attribution, such as: “*Display the source code link on top of the code after accepting*” (R63, 5★), and “*Let us know where this is giving the answer from*” (R144, 5★). Choudhuri et al. [74] also discuss developers’ trust towards GenAI.

Implication 5: Continuous user engagement and market awareness are key. Active communication helps assistant developers gather feedback and improve user satisfaction. For example, one user highlight the value of direct engagement: “Join the Discord channel and get your questions answered, feature requests responded to, and see all the cool stuff the open source community is coming up with!” (R340, 5★). Studying competitors, similar to practices in mobile app development [75], helps identify standard features and differentiation opportunities. Users favor assistants that offer unique capabilities beyond what others provide. Another good example is Cursor, which actively communicates with and hears from its users via various channels.

B. How much do the extensions vary in terms of capabilities?

To better understand the diversity of AI coding assistants, we conduct a small-scale analysis by manually reviewing the descriptions and documentation of the 32 extensions analyzed in our study. Based on their capabilities, we categorize them into three groups: (1) extensions limited to line-level code completion (e.g., tab-to-complete); (2) extensions with chat-based functionalities, such as bug fixing or documentation generation; and (3) extensions supporting other tasks.

Among the extensions, 5 offer only line-level completion, 18 support both chat-based and line-level features, 5 offer chat capabilities alone, and 4 focus on other tasks such as documentation generation, code review, or code snippet management. Further analysis shows that 63% of the reviews for chat-based extensions are positive, with users focused on functionality (33.8%) and broader discussions (18.7% under “Other”). Line-level tools display a near-even split between likes (51%) and dislikes (49%), with user comments often centering on performance (12.8%) and comparisons with other tools (12.8%). Other extensions are the most positively received (77% like) and are heavily praised for usability (41.9%).

C. Threats to Validity

Internal. Two annotators conduct all coding and sentiment analysis. Although inter-rater agreement is high in the final round, their shared background may introduce confirmation bias. Early collaborative rounds and evolving definitions risk “locking in” initial interpretations. To mitigate this, we delay discussion until after independent blind coding. While seeding five top-level categories may bias consolidation toward those dimensions, annotators retain full flexibility to revise or add categories and subcategories in every round.

External. Our study focuses on the VS Code Marketplace, which may raise concerns about generalizability to other development environments. However, VS Code is currently the most widely used IDE, with over 74% of developers reporting it as their primary editor. Its widespread adoption and active extension ecosystem make it a strong proxy for understanding mainstream developer interactions with AI coding assistants. While some platform-specific differences may exist, future work could explore whether the observed patterns hold in other environments such as JetBrains IDEs or cloud-based tools.

VIII. CONCLUSION AND FUTURE WORK

In this study, we analyze 361 reviews of 32 popular assistants. Using a hybrid card-sorting approach, we develop a three-level taxonomy with 8 categories, 16 subcategories, and 62 labels. Suggestion quality is paramount—accuracy is praised, while redundancy, incompleteness draw criticism. A major challenge is context awareness: assistants can interpret code but often fail to retrieve or retain relevant context. Usability also matters and high resource usage remains a concern. We also outline five implications to support the development of more effective AI coding assistants.

Future research could automate the labeling process using the taxonomy from this study and sentiment assessment to enable large-scale quantitative analysis. A key challenge here is developing LLM-based methods that reliably detect and classify emerging labels or categories while minimizing human-in-the-loop effort, including automating the hybrid card-sorting process in iterative taxonomy refinement.

In addition, expanding data sources beyond the VS Code Marketplace to platforms like Reddit and X (formerly Twitter) could provide complementary perspectives and richer insights into user perceptions. Another promising direction is to differentiate between types of AI coding assistants. For instance, future work could compare line-level code completion and chat-based assistants, and, through automated labeling, investigate how users perceive these tools differently. Replication package at <https://figshare.com/s/47c5ce42fd7e1dee69fe>.

ACKNOWLEDGMENTS

This research is supported by the Ministry of Education, Singapore under its Academic Research Fund Tier 3 (Award ID: MOET32020-0004). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

REFERENCES

- [1] H. Mozannar, G. Bansal, A. Fourney, and E. Horvitz, “Reading between the lines: Modeling user behavior and costs in ai-assisted programming,” ser. CHI ’24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: <https://doi.org/10.1145/3613904.3641936>
- [2] A. Sergeyuk, Y. Golubev, T. Bryksin, and I. Ahmed, “Using ai-based coding assistants in practice: State of affairs, perceptions, and ways forward,” *Information and Software Technology*, vol. 178, 2025.
- [3] J. Chen, X. Hu, Z. Li, C. Gao, X. Xia, and D. Lo, “Code search is all you need? improving code suggestions with code search,” in *Proceedings of the IEEE/ACM 46th international conference on software engineering*, 2024, pp. 1–13.
- [4] B. Yang, H. Tian, J. Ren, H. Zhang, J. Klein, T. Bissyande, C. Le Goues, and S. Jin, “Morepair: Teaching llms to repair code via multi-objective fine-tuning,” *ACM Trans. Softw. Eng. Methodol.*, May 2025, just Accepted. [Online]. Available: <https://doi.org/10.1145/3735129>
- [5] J. T. Liang, C. Yang, and B. A. Myers, “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.
- [6] L. Wilkinson, “Github copilot drives revenue growth amid subscriber base expansion,” 2024, accessed: Feb 28, 2025. [Online]. Available: <https://www.ciodive.com/news/github-copilot-subscriber-count-revenue-growth/706201/>
- [7] A. M. McNutt, C. Wang, R. A. Deline, and S. M. Drucker, “On the design of ai-powered code assistants for notebooks,” in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: <https://doi.org/10.1145/3544548.3580940>
- [8] C. Liu, X. Zhang, H. Zhang, Z. Wan, Z. Huang, and M. Yan, “An empirical study of code search in intelligent coding assistant: Perceptions, expectations, and directions,” in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, 2024, pp. 283–293.
- [9] J. Prather, B. N. Reeves, P. Denny, B. A. Becker, J. Leinonen, A. Luxton-Reilly, G. Powell, J. Finnie-Ansley, and E. A. Santos, “‘it’s weird that it knows what i want’: Usability and interactions with copilot for novice programmers,” *ACM Trans. Comput.-Hum. Interact.*, vol. 31, no. 1, Nov. 2023. [Online]. Available: <https://doi.org/10.1145/3617367>
- [10] C. Wang, J. Hu, C. Gao, Y. Jin, T. Xie, H. Huang, Z. Lei, and Y. Deng, “How practitioners expect code completion?” in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 1294–1306. [Online]. Available: <https://doi.org/10.1145/3611643.3616280>
- [11] P. Vaithilingam, T. Zhang, and E. L. Glassman, “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.
- [12] S. Barke, M. B. James, and N. Polikarpova, “Grounded copilot: How programmers interact with code-generating models,” *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA1, Apr. 2023. [Online]. Available: <https://doi.org/10.1145/3586030>
- [13] M. C. Davis, E. Aghayi, T. D. Latoza, X. Wang, B. A. Myers, and J. Sunshine, “What’s (not) working in programmer user studies?” *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 5, pp. 1–32, 2023.
- [14] A. Sergeyuk, I. Zakharov, E. Koshchenko, and M. Izadi, “Human-ai experience in integrated development environments: A systematic literature review,” *arXiv preprint arXiv:2503.06195*, 2025.
- [15] D. Pagano and W. Maalej, “User feedback in the appstore: An empirical study,” in *2013 21st IEEE International Requirements Engineering Conference (RE)*, 2013, pp. 125–134.
- [16] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, “A survey of app store analysis for software engineering,” *IEEE transactions on software engineering*, vol. 43, no. 9, pp. 817–847, 2016.
- [17] A. A. Al-Subaihini, F. Sarro, S. Black, L. Capra, and M. Harman, “App store effects on software engineering practices,” *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 300–319, 2019.

- [18] B. Zhang, P. Liang, X. Zhou, A. Ahmad, and M. Waseem, "Demystifying practices, challenges and expected features of using github copilot," *International Journal of Software Engineering and Knowledge Engineering*, vol. 33, no. 11n12, pp. 1653–1672, 2023.
- [19] X. Zhou, P. Liang, B. Zhang, Z. Li, A. Ahmad, M. Shahin, and M. Waseem, "Exploring the problems, their causes and solutions of ai pair programming: A study on github and stack overflow," *Journal of Systems and Software*, vol. 219, p. 112204, 2025.
- [20] J. Liao, G. Yang, D. Kavalier, V. Filkov, and P. Devanbu, "Status, identity, and language: A study of issue discussions in github," *PLoS one*, vol. 14, no. 6, p. e0215059, 2019.
- [21] D. Ford, J. Smith, P. J. Guo, and C. Parnin, "Paradise unplugged: Identifying barriers for female participation on stack overflow," in *Proceedings of the 2016 24th ACM SIGSOFT International symposium on foundations of software engineering*, 2016, pp. 846–857.
- [22] Stack Overflow, "Stack Overflow Developer Survey 2024," Online, 2024, available: <https://survey.stackoverflow.co/2024/>.
- [23] Y. Liu, C. Tantithamthavorn, and L. Li, "Protect Your Secrets: Understanding and Measuring Data Exposure in VSCode Extensions," in *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Los Alamitos, CA, USA: IEEE Computer Society, Mar. 2025, pp. 551–562.
- [24] Microsoft, "Visual Studio Marketplace," 2025, accessed: February 21, 2025. [Online]. Available: <https://marketplace.visualstudio.com>
- [25] J. Dąbrowski, E. Letier, A. Perini, and A. Susi, "Analysing app reviews for software engineering: a systematic literature review," *Empirical Softw. Engg.*, vol. 27, no. 2, Mar. 2022. [Online]. Available: <https://doi.org/10.1007/s10664-021-10065-7>
- [26] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1110–1121. [Online]. Available: <https://doi.org/10.1145/3377811.3380395>
- [27] L. Y. Conrad and V. M. Tucker, "Making it tangible: hybrid card sorting within qualitative interviews," *Journal of Documentation*, vol. 75, no. 2, pp. 397–416, Jan 2019. [Online]. Available: <https://doi.org/10.1108/JD-06-2018-0091>
- [28] GitHub, "Github copilot: The agent awakens," 2025, accessed: February 16 2025. [Online]. Available: <https://github.blog/news-insights/product-news/github-copilot-the-agent-awakens/>
- [29] Z. Yang, Z. Sun, T. Z. Yue, P. Devanbu, and D. Lo, "Robustness, security, privacy, explainability, efficiency, and usability of large language models for code," *arXiv preprint arXiv:2403.07506*, 2024.
- [30] GitHub, "Survey Reveals AI's Impact on the Developer Experience," <https://github.blog/news-insights/research/survey-reveals-ais-impact-on-the-developer-experience/>, 2024, accessed: May 25, 2025.
- [31] A. Ziegler, E. Kalliamvakou, X. A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, and E. Aftandilian, "Productivity assessment of neural code completion," in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, ser. MAPS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 21–29. [Online]. Available: <https://doi.org/10.1145/3520312.3534864>
- [32] A. Shah, A. Chernova, E. Tomson, L. Porter, W. G. Griswold, and A. G. Soosai Raj, "Students' use of github copilot for working with large code bases," in *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSETS 2025. New York, NY, USA: Association for Computing Machinery, 2025, p. 1050–1056. [Online]. Available: <https://doi.org/10.1145/3641554.3701800>
- [33] M. V. Phong, T. T. Nguyen, H. V. Pham, and T. T. Nguyen, "Mining user opinions in mobile app reviews: A keyword-based approach (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 749–759.
- [34] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, "Arminer: mining informative reviews for developers from mobile app marketplace," ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 767–778. [Online]. Available: <https://doi.org/10.1145/2568225.2568263>
- [35] E. Guzman and W. Maalej, "How do users like this feature? a fine grained sentiment analysis of app reviews," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, 2014, pp. 153–162.
- [36] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Requirements Engineering*, vol. 21, no. 3, pp. 311–331, Sep 2016. [Online]. Available: <https://doi.org/10.1007/s00766-016-0251-9>
- [37] N. Jha and A. Mahmoud, "Mining non-functional requirements from app store reviews," *Empirical Software Engineering*, vol. 24, no. 6, pp. 3659–3695, Dec 2019. [Online]. Available: <https://doi.org/10.1007/s10664-019-09716-7>
- [38] GitHub, "GitHub Copilot," <https://marketplace.visualstudio.com/items?itemName=GitHub.copilot>, 2025, retrieved Feb 22, 2025.
- [39] Visual Studio | Marketplace, "VS Code Marketplace: AI extensions search results," <https://marketplace.visualstudio.com/search?term=AI&target=VSCode&category=All%20categories&sortBy=Installs>, 2025, retrieved Feb 21, 2025.
- [40] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, Dec. 2024.
- [41] SurveyMonkey, "Sample size calculator," 2025, accessed: Feb 28, 2025. [Online]. Available: <https://www.surveymonkey.com/mp/sample-size-calculator/>
- [42] H. Hata, C. Treude, R. G. Kula, and T. Ishio, "9.6 million links in source code comments: Purpose, evolution, and decay," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1211–1221.
- [43] Y. Lyu, H. J. Kang, R. Widyasari, J. Lawall, and D. Lo, "Evaluating szz implementations: An empirical study on the linux kernel," *IEEE Trans. Softw. Eng.*, vol. 50, no. 9, p. 2219–2239, Sep. 2024. [Online]. Available: <https://doi.org/10.1109/TSE.2024.3406718>
- [44] J. Wang, L. Li, and A. Zeller, "Restoring execution environments of jupyter notebooks," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1622–1633.
- [45] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.
- [46] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.
- [47] P. E. McKnight and J. Najab, "Mann-whitney u test," *The Corsini encyclopedia of psychology*, pp. 1–1, 2010.
- [48] "Variability and mutability, contribution to the study of statistical distributions and relations," *Studi Economico-Giuridici della R. Universita de Cagliari*, 1912.
- [49] Z. Kurtanović and W. Maalej, "Mining user rationale from software reviews," in *2017 IEEE 25th international requirements engineering conference (RE)*. IEEE, 2017, pp. 61–70.
- [50] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.
- [51] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of ai on developer productivity: Evidence from github copilot," *arXiv preprint arXiv:2302.06590*, 2023.
- [52] B. Martinović and R. Rozić, "Perceived impact of ai-based tooling on software development code quality," *SN Computer Science*, vol. 6, no. 1, p. 63, Jan 2025. [Online]. Available: <https://doi.org/10.1007/s42979-024-03608-4>
- [53] S. Imai, "Is github copilot a substitute for human pair-programming? an empirical study," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 319–321. [Online]. Available: <https://doi.org/10.1145/3510454.3522684>
- [54] A. Moradi Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, and Z. M. J. Jiang, "Github copilot ai pair programmer: Asset or liability?" *Journal of Systems and Software*, vol. 203, p. 111734, 2023.
- [55] J. Prather, B. N. Reeves, P. Denny, B. A. Becker, J. Leinonen, A. Luxton-Reilly, G. Powell, J. Finnie-Ansley, and E. A. Santos, "'it's weird that it knows what i want': Usability and interactions with copilot for novice programmers," *ACM Trans. Comput.-Hum. Interact.*, vol. 31, no. 1, Nov. 2023. [Online]. Available: <https://doi.org/10.1145/3617367>
- [56] Z. Yang, B. Xu, J. M. Zhang, H. J. Kang, J. Shi, J. He, and D. Lo, "Stealthy backdoor attack for code models," *IEEE Transactions on Software Engineering*, vol. 50, no. 4, pp. 721–741, 2024.
- [57] Z. Yang, Z. Zhao, C. Wang, J. Shi, D. Kim, D. Han, and D. Lo, "Unveiling memorization in code models," ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: <https://doi.org/10.1145/3597503.3639074>

- [58] D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, and B. Myers, "Using an llm to help with code understanding," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: ACM, 2024.
- [59] Z. Sun, X. Du, F. Song, S. Wang, and L. Li, "When neural code completion models size up the situation: Attaining cheaper and faster completion through dynamic model inference," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. Association for Computing Machinery, 2024.
- [60] T. Ge, J. Hu, L. Wang, X. Wang, S.-Q. Chen, and F. Wei, "In-context autoencoder for context compression in a large language model," *arXiv preprint arXiv:2307.06945*, 2023.
- [61] G. Pinto, C. De Souza, J. B. Neto, A. Souza, T. Gotto, and E. Monteiro, "Lessons from building stackspot ai: A contextualized ai coding assistant," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 408–417. [Online]. Available: <https://doi.org/10.1145/3639477.3639751>
- [62] W. Luo, J. W. Keung, B. Yang, J. Klein, T. F. Bissyande, H. Tian, and B. Le, "Unlocking llm repair capabilities in low-resource programming languages through cross-language translation and multi-agent refinement," *arXiv preprint arXiv:2503.22512*, 2025.
- [63] W. Luo, J. Keung, B. Yang, H. Ye, C. Le Goues, T. F. Bissyandé, H. Tian, and X. B. D. Le, "When fine-tuning llms meets data privacy: An empirical study of federated learning in llm-based program repair," *ACM Trans. Softw. Eng. Methodol.*, May 2025, just Accepted. [Online]. Available: <https://doi.org/10.1145/3733599>
- [64] H. Mozannar, G. Bansal, A. Fourney, and E. Horvitz, "When to show a suggestion? integrating human feedback in ai-assisted programming," ser. AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. [Online]. Available: <https://doi.org/10.1609/aaai.v38i9.28878>
- [65] Z. Sun, X. Du, F. Song, S. Wang, M. Ni, and L. Li, "Don't complete it! preventing unhelpful code completion for productive and sustainable neural code completion systems," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2023, pp. 324–325.
- [66] J. Shi, Z. Yang, B. Xu, H. J. Kang, and D. Lo, "Compressing pre-trained models of code into 3 mb," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: <https://doi.org/10.1145/3551349.3556964>
- [67] J. Shi, Z. Yang, H. J. Kang, B. Xu, J. He, and D. Lo, "Greening large language models of code," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Society*, 2024, pp. 142–153.
- [68] J. Shi, Z. Yang, and D. Lo, "Efficient and green large language models for software engineering: Vision and the road ahead," *ACM Transactions on Software Engineering and Methodology*, 2024.
- [69] V. Mittal, W. T. Ross Jr., and P. M. Baldasare, "The asymmetric impact of negative and positive attribute-level performance on overall satisfaction and repurchase intentions," *Journal of Marketing*, vol. 62, no. 1, pp. 33–47, 1998.
- [70] B. B. Holloway and S. E. Beatty, "Satisfiers and dissatisfiers in the online environment: A critical incident assessment," *Journal of Service Research*, vol. 10, no. 4, pp. 347–364, 2008.
- [71] Z. Yang, J. Shi, J. He, and D. Lo, "Natural attack for pre-trained models of code," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1482–1493. [Online]. Available: <https://doi.org/10.1145/3510003.3510146>
- [72] Y. Lyu, T. Le-Cong, H. J. Kang, R. Widyasari, Z. Zhao, X.-B. D. Le, M. Li, and D. Lo, "Chronos: Time-aware zero-shot identification of libraries from vulnerability reports," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23. IEEE Press, 2023, p. 1033–1045.
- [73] Y. Lyu, Z. Yang, Y. Niu, J. Jiang, and D. Lo, "Do existing testing tools really uncover gender bias in text-to-image models?" *arXiv preprint arXiv:2501.15775*, 2025.
- [74] R. Choudhuri, B. Trinkenreich, R. Pandita, E. Kalliamvakou, I. Steinmacher, M. Gerosa, C. Sanchez, and A. Sarma, "What guides our choices? modeling developers' trust and behavioral intentions towards genai," *arXiv preprint arXiv:2409.04099*, 2024.
- [75] A. A. Al-Subaihini, F. Sarro, S. Black, L. Capra, and M. Harman, "App store effects on software engineering practices," *IEEE Transactions on Software Engineering*, vol. 47, no. 2, pp. 300–319, 2021.