Exploring the Problems, their Causes and Solutions of AI Pair Programming: A Study on GitHub and Stack Overflow

Xiyu Zhou[a], Peng Liang[a,*], Beiqi Zhang[a], Zengyang Li[b], Aakash Ahmad[c], Mojtaba Shahin[d] and Muhammad Waseem[e]

[a]*School of Computer Science, Wuhan University, Wuhan, China*
[b]*School of Computer Science, Central China Normal University, Wuhan, China*
[c]*School of Computing and Communications, Lancaster University Leipzig, Leipzig, Germany*
[d]*School of Computing Technologies, RMIT University, Melbourne, Australia*
[e]*Faculty of Information Technology, University of Jyväskylä, Jyväskylä, Finland*

ARTICLE INFO

ABSTRACT

With the recent advancement of Artificial Intelligence (AI) and Large Language Models (LLMs), AI-based code generation tools become a practical solution for software development. GitHub Copilot, the AI pair programmer, utilizes machine learning models trained on a large corpus of code snippets to generate code suggestions using natural language processing. Despite its popularity in software development, there is limited empirical evidence on the actual experiences of practitioners who work with Copilot. To this end, we conducted an empirical study to understand the problems that practitioners face when using Copilot, as well as their underlying causes and potential solutions. We collected data from 473 GitHub issues, 706 GitHub discussions, and 142 Stack Overflow posts. Our results reveal that (1) *Operation Issue* and *Compatibility Issue* are the most common problems faced by Copilot users, (2) *Copilot Internal Error*, *Network Connection Error*, and *Editor/IDE Compatibility Issue* are identified as the most frequent causes, and (3) *Bug Fixed by Copilot*, *Modify Configuration/Setting*, and *Use Suitable Version* are the predominant solutions. Based on the results, we discuss the potential areas of Copilot for enhancement, and provide the implications for the Copilot users, the Copilot team, and researchers.

1. Introduction

In software development, developers strive to achieve automation and intelligence to generate most code automatically with minimal human coding effort. Several studies (e.g., Luan et al. (2019), Robillard et al. (2010)) and software products (e.g., Eclipse Code Recommenders (Eclipse, 2019)) have been dedicated to improving the efficiency of developers through the development of systems that can recommend and generate code. However, early AI code generation tools mostly used heuristic rules or expert systems, which faced significant limitations, including inflexibility and challenges in scalability (Jiang et al., 2024), making them less adaptable to varying contexts. Large Language Models (LLMs) are a type of natural language processing (NLP) technique based on deep learning that is capable of automatically learning the grammar, semantics, and pragmatics of language, and generating a wide variety of contents. Due to the extensive number of parameters and large-scale training datasets, LLMs have demonstrated powerful capabilities in NLP, often approaching or even surpassing human-level performance in NLP tasks such as text translation (Huang et al., 2023) and sentiment analysis (Amin et al., 2023) (Chen et al., 2023). The expansion of model parameters also enables LLMs to capture intricate language patterns, resulting in sustained performance enhancements across a wide range of downstream tasks (Wei et al., 2022). Therefore, the emergence of LLMs has significantly re-shaped the field of code generation tasks (Chen et al., 2021). Recently, AI code generation tools driven by LLMs that have been trained on large amounts of code snippets are increasingly in the spotlight (e.g., AI-augmented development in Gartner Trends 2024 (Gartner, 2023)), making it possible for programmers to automatically generate code with minimized human effort (Austin et al., 2021).

On June 29, 2021, GitHub and OpenAI jointly announced the launch of a new product named GitHub Copilot (GitHub, 2024d). This innovative tool is powered by OpenAI's Codex, a large-scale neural network model that is trained on a massive dataset of source code and natural language text. The goal of GitHub Copilot is to provide advanced code autocompletion and generation capabilities to developers, effectively acting as an "AI pair programmer" that can assist with coding tasks in real-time. Copilot has been designed to work with a wide range of Integrated Development Environments (IDEs) and code editors, such as VSCode, Visual Studio, Neovim, and JetBrains (GitHub, 2024d). By collecting contextual information like function names and comments, Copilot is able to generate code snippets in a variety of programming languages (e.g., Python, C++, Java), which can improve developers' productivity and help them complete coding tasks more efficiently (Imai, 2022).

*Corresponding author.

✉ xiyuzhou@whu.edu.cn (X. Zhou); liangp@whu.edu.cn (P. Liang); zhangbeiqi@whu.edu.cn (B. Zhang); zengyangli@ccnu.edu.cn (Z. Li); ahmad.aakash@gmail.com (A. Ahmad); mojtaba.shahin@rmit.edu.au (M. Shahin); muhammad.m.waseem@jyu.fi (M. Waseem)

ORCID(s):

Since its release, Copilot has gained significant attention within the developer community, and it had a total of 1.3 million paid users till Feb of 2024 (Wilkinson, 2024). Many studies identify the effectiveness and concerns about the potential impact on code security and intellectual property (Pearce et al., 2022) (Bird et al., 2023) (Jaworski and Piotrkowski, 2023). Some prior research investigated the quality of the code generated by Copilot (Yetistiren et al., 2022) (Nguyen and Nadi, 2022), while others examined its performance in practical software development (Imai, 2022) (Barke et al., 2023) (Peng et al., 2023).

However, there is currently a lack of systematic categorization of the problems arise during the practical use of Copilot from the perspective of developers, as well as the causes behind them and solutions for addressing them. To this end, we conducted a thorough analysis of the problems faced by software developers when coding with GitHub Copilot, as well as their causes and solutions, by collecting data from GitHub Issues, GitHub Discussions, and Stack Overflow (SO) posts, which would help to understand the limitations of Copilot in practical settings.

Our findings show that: (1) *Operation Issue* and *Compatibility Issue* are the most common problems faced by developers, (2) *Copilot Internal Error*, *Network Connection Error*, and *Editor/IDE Compatibility Issue* are identified as the most frequent causes, and (3) *Bug Fixed by Copilot*, *Modify Configuration/Setting*, and *Use Suitable Version* are the predominant solutions.

The contributions of this work are that:

- We provided a two-level taxonomy for the problems of using Copilot in the software development practice, and a one-level taxonomy for the causes of the problems and the solutions to address the problems.

- We drew a mapping from the identified problems to their causes and solutions.

- We proposed practical guidelines for Copilot users, the Copilot team, and other researchers.

The rest of this paper is structured as follows: Section 2 presents the Research Questions (RQs) and research process. Section 3 provides the results and their interpretation. Section 4 discusses the implications based on the research results. Section 5 clarifies the potential threats to the validity of this study. Section 6 reviews the related work. Finally, Section 7 concludes this work along with the future directions.

2. Methodology

The research goal of this study is to identify the problems encountered by users when coding with Copilot, as well as their underlying causes and potential solutions. We formulated three RQs presented in Section 2.1. Fig. 1 provides the overview of the research process.

2.1. Research Questions

RQ1: What are the problems faced by users while using Copilot in software development practice?
Rationale: GitHub Copilot is one of the most popular AI-assisted coding tools and has been widely used for software development with 1.3 million paid users till Feb of 2024 (Wilkinson, 2024). Consequently, it is important to understand the specific challenges and problems users face while using this tool in software development practice.

RQ2: What are the underlying causes of these problems?
Rationale: Understanding the causes of the problems identified in RQ1 is essential to developing effective solutions to address them. By identifying these causes, this study can provide insights into how to improve the design and functionality of Copilot.

RQ3: What are the potential solutions to address these problems?
Rationale: Exploring the solutions of the problems identified in RQ1 and the causes identified in RQ2 is essential to improving the user experience when using Copilot. By identifying these solutions, this study can gain insight into potential improvements that enhance the functionality and usability of Copilot.

2.2. Data Collection

We collected data from three sources: GitHub Issues[1], GitHub Discussions[2], and SO posts[3]. Data from these three sources has been frequently used in empirical studies to gather various developer-reported problems in software engineering (Bird et al., 2023) (de Dieu et al., 2023) (El aoun et al., 2021). GitHub Issues is a commonly used feature on GitHub for tracking bugs and feature requests and reporting other issues related to software projects. It allows us to capture the specific problems that users have encountered when coding with Copilot. GitHub Discussions is a feature that allows open-ended discussions among project contributors and community members. It offers a central hub for project-related discussions and knowledge sharing. Stack Overflow is a public Q&A platform covering a wide range of programming, development, and technology topics, also including inquiries about using Copilot.

Considering that Copilot was announced and started its technical preview on June 29, 2021, we chose to collect the data that were created after that date. The data collection was conducted on June 18, 2023. To answer RQ3, i.e., the solutions for addressing Copilot usage problems, we chose to collect closed GitHub issues, as well as answered GitHub discussions and SO posts. Specifically, for GitHub issues, we used "Copilot" as the keyword to search closed Copilot-related issues globally in the entire GitHub, and a total of 4,057 issues were retrieved. We also employed "Copilot" as a keyword to search answered posts in SO, resulting in 679 retrieved posts. Note that we did not use the "Copilot" tag

---

[1] https://docs.github.com/en/issues
[2] https://github.com/orgs/community/discussions/categories/copilot
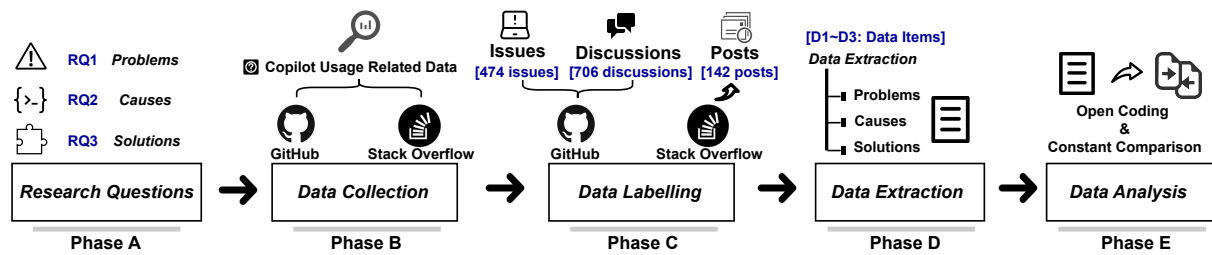[3] https://stackoverflow.com/

Figure 1: Overview of the research process

for retrieval because the keyword-based method allows us to obtain a more exhaustive dataset. Different from GitHub issues and SO posts, GitHub discussions related to Copilot are grouped under the "Copilot" subcategory within the "Product" category in the GitHub Community, which is recommended by the Copilot team to provide feedback on Copilot usage. Therefore, the discussions within the "Copilot" subcategory frequently receive attention and responses from the Copilot team, making them highly valuable for answering RQ2 and RQ3 in our research. Given the high relevance of these discussions to Copilot usage, we collected all the 925 answered discussions under the "Copilot" subcategory.

2.3. Data Labelling

We conducted the data labelling on the collected data to filter out those cannot be used for this study. The criteria for data filtering is as follows: the issue, discussion, or post should contain specific information related to the use of GitHub Copilot.

2.3.1. Pilot Data Labelling

To minimize personal bias in the formal labelling process, the first and third authors conducted a pilot data labelling. For GitHub issues and discussions, we randomly selected 100 and 25 from each, making up 2.5% of the total count. Due to the small quantity of SO posts, we randomly selected 35, which constitutes 5% of the total posts. Selecting a certain proportion of data from different platforms respectively is to verify whether the criteria of the two authors are consistent across various data sources. The inter-rater reliability between the two authors was measured by the Cohen's Kappa coefficient (Cohen, 1960), resulting in values of 0.824, 0.834, and 0.806, which indicate a reasonable level of agreement between the two authors. For any discrepancies in the results, the two authors engaged in discussions with the second author to reach a consensus. To be specific, the first author and the third author presented their reasons for why the data should or should not be included in our research. Then, the three authors analyzed and discussed these reasons according to the criteria for data filtering, ultimately arriving at a conclusion accepted by the three authors. The results of pilot data labelling were compiled and recorded in MS Excel (Zhou et al., 2024).

2.3.2. Formal Data Labelling

The first and third authors then conducted the formal data labelling. During this process, we excluded a large amount of data not related to our research. For instance, "Copilot" may refer to other meanings in some situations, such as the "co-pilot" of an aircraft. Additionally, Copilot might be mentioned in a straightforward manner without additional information, like a post mentioned, "*You can try using Copilot, which is amazing*". We also excluded such cases of data since they could not provide useful information about the usage of Copilot. During the labelling process, any result on which the two authors disagreed was subject to discussion with the second author until an agreement was reached. Ultimately, the two authors collected 473 GitHub issues, 706 GitHub discussions, and 142 SO posts. The data labelling results were compiled and recorded in MS Excel (Zhou et al., 2024).

2.4. Data Extraction

To answer the three RQs in Section 2.1, we established a set of data items for data extraction, as presented in Table 1. Data items D1-D3 intend to extract the information of problems, underlying causes, and potential solutions from the filtered data to answer RQ1-RQ3, respectively. These three data items could be extracted from any part of a GitHub issue, discussion, or SO post, such as the title, the problem description, comments, and discussions.

2.4.1. Pilot Data Extraction

The first and third author conducted a pilot data extraction on 20 randomly selected GitHub issues, 20 discussions, and 20 SO posts, and in case of any discrepancies, the second author was involved to reach a consensus. The results indicated that the three data items could be extracted from our dataset. Based on the observation, we established the following criteria for formal data extraction: (1) If the same problem was identified by multiple users, we recorded it only once. (2) If multiple problems were identified within the same GitHub issue, GitHub discussion, or SO post, we recorded each one separately. (3) For a problem that has multiple causes mentioned, we only recorded the cause confirmed by the reporter of the problem or the Copilot team as the root cause. (4) For a problem that has multiple solutions suggested, we only recorded the solutions that were
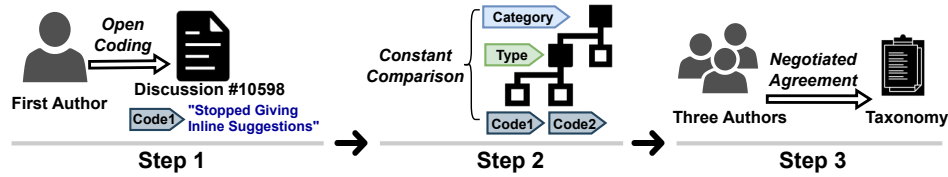
Figure 2: The process of data analysis

Table 1

Data items extracted and their corresponding RQs

#	Data Item	Description	RQ
D1	Problem	*The key point(s) of the problem from GitHub issues, GitHub discussions, and SO posts*	RQ1
D2	Cause	*The key point(s) of the cause from GitHub issues, GitHub discussions, and SO posts*	RQ2
D3	Solution	*The key point(s) of the solution from GitHub issues, GitHub discussions, and SO posts*	RQ3

confirmed by the reporter of the problem or the Copilot team to actually solve the problem.

2.4.2. Formal Data Extraction

The first and third authors conducted the formal data extraction from the filtered dataset to extract the data items. Subsequently, they discussed and reached a consensus with the second author on inconsistencies to ensure that the data extraction process adhered to the predetermined criteria. Each extracted data item was reviewed multiple times by the three authors to ensure accuracy. The data extraction results were compiled and recorded in MS Excel (Zhou et al., 2024).

Although we selected closed GitHub issues, and answered GitHub discussions and SO posts, not all collected data includes the cause and solution of a problem. Sometimes, the respondents to a Copilot related problem might offer a solution without detailed analysis, preventing us from extracting the underlying causes. In other situations, although the cause of a problem was identified, the user did not describe the specific resolution process. For example, a user found that Copilot "*cannot work correctly on VSCode remote server*" and realized it was due to "*the bad network*", but did not provide any solutions (Discussion #14907). Additionally, even when some responses provided both causes and solutions, they might not be accepted or proven effective by the problem's reporter or the Copilot team. For example, a user asked for "*a way to set up GitHub copilot in Google Colab*", but the user neither accepted nor replied to the three proposed answers (SO #72431032). Therefore, we cannot consider any of the three answers as an effective solution to his problem.

2.5. Data Analysis

To answer the three RQs formulated in Section 2, we conducted data analysis by using the Open Coding and Constant Comparison methods, which are two widely employed techniques from Grounded Theory during qualitative data analysis (Stol et al., 2016). Open Coding is a research method that encourages researchers to generate codes based on the actual content of the data rather than being restricted by pre-existing theoretical frameworks. These codes constitute descriptive summarizations of the data, aiming to capture the underlying themes. For Constant Comparison, researchers continuously compare the coded data, dynamically refining and adjusting the categories based on their similarities and differences before forming the final classification results.

As presented in Fig 2, the specific process of data analysis includes three steps: 1) The first author reviewed the collected data and then assigned descriptive codes that succinctly encapsulated the core themes. For instance, the issue in Discussion #10598 was coded as "*Stopped Giving Inline Suggestions*", which was reported by a user whose Copilot suddenly stopped providing code suggestions in VSCode. 2) The first author compared different codes to identify patterns, commonalities, and distinctions among them. Through this iterative comparison process, similar codes were merged into higher-level types and categories. For example, the code of Discussion #10598, along with other akin codes, formed into the type of FUNCTIONALITY FAILURE, which further belongs to the category of *Operation Issue*. Once uncertainties arose, the first author engaged in discussions with the second and third authors to achieve a consensus. Due to the nature of Constant Comparison, both the types and the categories were refined multiple times before reaching their final form. 3) The initial version of the analysis results was further verified by the second and third authors, and the negotiated agreement approach (Campbell et al., 2013) was employed to address the conflicts. The final results are presented in Section 3.

3. Results and Interpretation

In this section, we report the results of three RQs and provide their interpretation. The results of Copilot usage problems are categorized into two levels: categories (e.g., *Suggestion Content Issue*) and types (e.g., LESS EFFICIENT SUGGESTION). Meanwhile, the results for causes and solutions are organized as types only (e.g., *Network Connection Error*). We also provide the mapping relationship of Copilot related problems to their causes and solutions. As mentioned in Section 2.4.2, only causes that were proven to lead to the problems and solutions that could resolve the problems were extracted and provided in the results. Therefore, not all problems have corresponding causes and solutions. It is worth noting that due to the rapid update of Copilot, some problems and feature requests raised by users have already
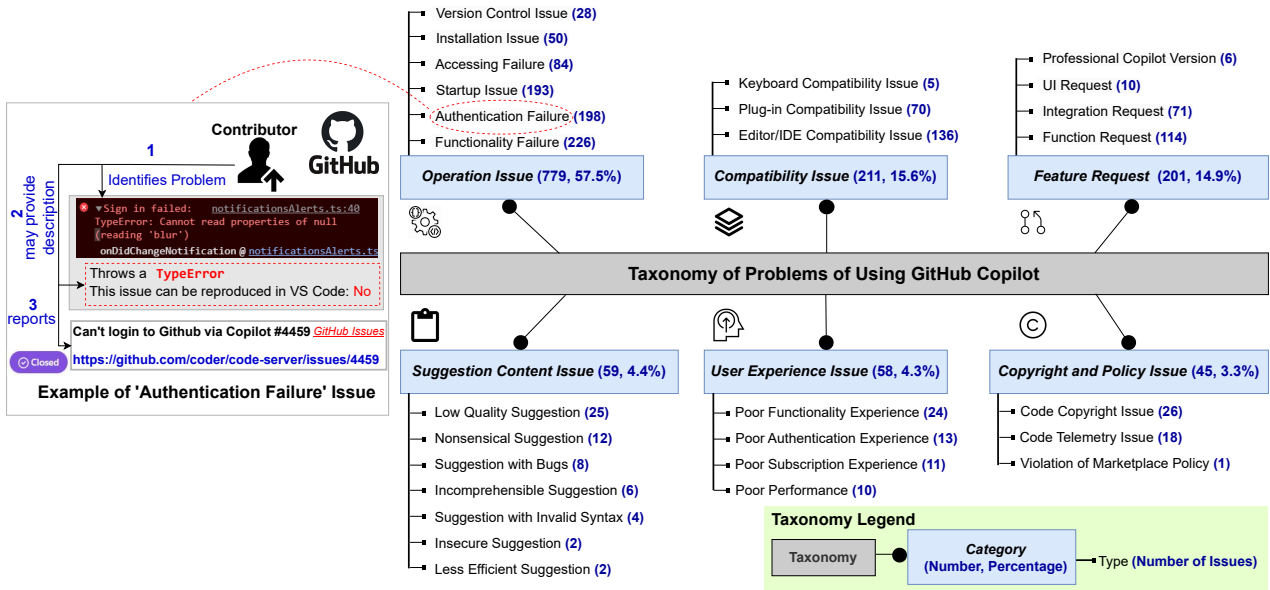
Figure 3: A taxonomy of problems when using GitHub Copilot

been addressed in newer releases of Copilot. We identified these two scenarios separately as two types of solutions, i.e., *Bug Fixed by Copilot* and *Feature Implemented by Copilot*. However, due to the absence of Copilot version information in the dataset, we could not consider the version information thoroughly in this study. To help understand the taxonomies of problems, causes, and solutions of using Copilot, we provide examples with the "#" symbol, which indicates the "GitHub Issue ID", "GitHub Discussion ID", or "SO Post ID" in the dataset (Zhou et al., 2024).

3.1. Type of Problems (RQ1)

Fig. 3 presents the taxonomy of the problems extracted from our dataset. We have identified a total of 1,355 problems related to Copilot usage. It can be observed that *Operation Issue* (57.5%) accounts for the majority of problems faced by Copilot users. Furthermore, there are a notable number of users who have countered *Compatibility Issue* (15.6%) when using Copilot in different environments, followed by users who have raised *Feature Request* (14.9%) based on their user experience and requirements. Additionally, smaller percentages were identified as *Suggestion Content Issue* (4.4%), *User Experience Issue* (4.3%), and *Copyright and Policy Issue* (3.3%).

3.1.1. Operation Issue (57.5%)

Operation Issue refers to a category of obstacles encountered by users when attempting to utilize some of the fundamental functions of Copilot. This category of problems is divided into six types, which are elaborated below.

- FUNCTIONALITY FAILURE refers to the abnormality of code generation-related features provided by Copilot. Copilot offers various interactive features to better

engage with users, such as "previous/next suggestion", "viewing all suggestions", and "configuration of shortcut keys to accept suggestions". Users may encounter exceptions when using these features. For example, a user reported that "*Copilot no longer suggesting on PyCharm*" after a period of not using it (Discussion #11199).

- STARTUP ISSUE refers to errors or malfunctions encountered by users attempting to run Copilot. This issue results in a complete failure of Copilot to execute and is typically accompanied by error messages. Such problems may arise either during a user's initial usage of Copilot or unexpectedly after several successful runs. For example, a user failed to activate Copilot after installing it on VSCode, and received an error message stating "*Cannot find module*" (Issue #380).

- AUTHENTICATION FAILURE refers to the issues related to user login and authentication difficulties when using Copilot. Copilot requires users to log in to their GitHub account before using the service. Only users with access permissions (including paid subscriptions, student identity verification, etc.) can use the code generation service of Copilot. During the authentication process, users may encounter AUTHENTICATION FAILURE, resulting in the inability to use Copilot. For example, a user was repeatedly prompted with the message "*waiting for GitHub Authentication*" in IDEA, and was unable to log in (SO #72505280).

- ACCESSING FAILURE refers to the situation where users fail to access Copilot's server, which often involves errors related to server connections. A user

may encounter an error message like "*GitHub Copilot could not connect to server*" (Discussion #11801).

- INSTALLATION ISSUE refers to the problems encountered by users during the installation process of Copilot, including installation errors, inability to find installation methods, and other related problems. For instance, a user failed to install Copilot on VSCode insiders, and the server log showed "*Error while installing the extension*" (Issue #187).

- VERSION CONTROL ISSUE refers to the problems that users encounter when adjusting the version of Copilot or its runtime environment (e.g., IDE), including the inability to upgrade the Copilot version or abnormal issues like continuing to prompt for upgrades even after upgrading. For example, a user reported that "*copilot plugin fails to update*" when using it in IntelliJ IDEA (Discussion #17298).

Interpretation: We identified *Operation Issue* at various stages of user interaction with Copilot. Users tend to report these problems and seek assistance, making *Operation Issue* the most prevalent category of problems related to Copilot. FUNCTIONALITY FAILURE (226), AUTHENTICATION FAILURE (198), and STARTUP ISSUE (193), are the top three types of such problems. We attribute the higher frequency of the first two types to the deficiencies in Copilot's feature design and stability, which are also influenced by users' environments in which Copilot operates. AUTHENTICATION FAILURE mainly stems from particular details encountered during the login process when users need to access Copilot with their GitHub accounts.

3.1.2. Compatibility Issue (15.6%)

This category covers the problems that arise from mismatches between Copilot and its runtime environment. Copilot operates as a plugin in various IDEs and code editors (e.g., VSCode and IntelliJ IDEA), and the complexity of the environments Copilot operates on can result in an increased number of compatibility issues. These problems are further classified into three types, which are elaborated below.

- EDITOR/IDE COMPATIBILITY ISSUE refers to issues arising from mismatches between Copilot and its IDE or editor. These problems typically manifest as Copilot being unable to operate properly in a specific IDE or editor. For example, a user previously found that Copilot "*does not work in Neovim*" while writing a Python program, even though the Copilot status showed that "*Copilot: Enabled and Online*" (SO #72174839).

- PLUG-IN COMPATIBILITY ISSUE refers to a type of matching issue that arises when Copilot and other plugins are activated and working together at the same time. Such problems can cause partial or complete malfunctions of Copilot and other plugins. They are usually identified through troubleshooting methods

such as disabling Copilot or other plugins. For instance, one user reported that "*a Keyboard shortcut conflict with Emmet*" prevented him from receiving code suggestions generated by Copilot (Issue #47).

- KEYBOARD COMPATIBILITY ISSUE refers to the situation when the functionality of Copilot can not be used in some uncommon keyboard layouts. For example, a user with a German keyboard layout could not use most of the code generation-related features of Copilot (Discussion #7094).

Interpretation: *Compatibility Issue* arises from the complex environments in which users utilize Copilot, as well as the compatibility robustness of Copilot itself. In the case of EDITOR/IDE COMPATIBILITY ISSUE (136), VSCode, the platform officially recommended for using Copilot, has garnered a higher number of reported issues about compatibility. We also identified many problems in other widely used IDEs, like Visual Studio, IntelliJ IDEA, and PyCharm. The appearance of PLUG-IN COMPATIBILITY ISSUE (70) is less predictable, which often arises when using Copilot with other code completion tools.

3.1.3. Feature Request (14.9%)

Feature Request refers to the features that users request to add or improve based on their experience and actual needs when using Copilot. These feature requests not only help improve the user experience of Copilot but also contribute to the exploration of how AI code generation tools like Copilot can better interact with developers. This category is further divided into four types, as shown below.

- FUNCTION REQUEST refers to the requests for developing new functions in Copilot, which typically arise from users' genuine needs and difficulties encountered while utilizing the tool. For example, a user requested that Copilot should be able to "*look at the context of a project with multiple files*", rather than generate code suggestions based on the context in a single file (SO #73848372).

- INTEGRATION REQUEST refers to a type of request for Copilot to be available on certain platforms or to be integrated with other plugins. This is mainly due to the desire of some users to use Copilot in the environments they are familiar with. For instance, a user called for "*Support for Intellij 2022.2 EAP family*" (Discussion #17045). The requests for integration also reflect the popularity of Copilot among developers to some extent.

- UI REQUEST refers to requests made by users for changes to the User Interface (UI) of Copilot, such as modifying the appearance of the Copilot icon. These requests generally aim to improve the visual effects and user experience of Copilot. For example, a user requested the addition of a "*status indicator*" to provide information about the current working status of Copilot (Issue #161).

- PROFESSIONAL COPILOT VERSION refers to requests from some users for a professional version of Copilot. These users are developers from certain companies who hope to receive more professional and reliable code generation services in their actual work. They may have higher requirements for the reliability and security of Copilot's code, as well as team certification and other aspects. For example, a user asked for "*an on-prem version for companies to purchase*", so that they could deploy Copilot in a local environment (Discussion #38858).

Interpretation: For FUNCTION REQUEST (114), we observed that users expressed a desire for greater flexibility in configuring Copilot to align more closely with their development habits. Common requests include the ability to accept Copilot's suggestions word by word and to specify where Copilot should automatically operate in terms of file types or code development scopes. More innovative demands involve the need for Copilot to provide suggestions according to the whole project, as well as features like code explanation and chat functionality which have been provided in GitHub Copilot Chat (Zhao, 2023). INTEGRATION REQUEST (71) reflects the wish of developers to use Copilot in their familiar environments. This places greater demands on the Copilot team, as we have identified a significant number of *Compatibility Issues*.

3.1.4. Suggestion Content Issue (4.4%)

This category of problems refers to the issues related to the content of the code generated by Copilot. The generation of code suggestions is the core feature of AI code generation tools like Copilot, and the quality of the suggestions directly determines whether users will adopt them. Therefore, the content of the generated code is naturally an area of concern for users, researchers, and the Copilot team. These problems are further divided into seven specific situations, which are detailed below.

- LOW QUALITY SUGGESTION refers to situations where Copilot is unable to comprehend the context sufficiently to generate useful code. Such code suggestions may not have any syntactical errors, but due to their poor quality, they are unlikely to be adopted by users. For instance, Copilot once generated an empty method containing only a return statement without meeting the requirements specified in the user's code (Discussion #6631).

- NONSENSICAL SUGGESTION refers to the code suggestions provided by Copilot that are completely irrelevant to the needs of users. Such suggestions are considered almost unusable and provide little heuristic assistance to the user. For example, a user once received an inaccessible fake URL generated by Copilot, which was of no help with his programming task (Discussion #14212).

- SUGGESTION WITH BUGS refers to the situation where Copilot is able to generate relevant code based on the context, but the suggested code contains some bugs. This can result in the program being able to run, but not as the developer intended, or in some cases, it may cause errors or crashes. For example, a user reported that Copilot suggested using "*setState(!state)*" instead of "*setState(true)*", which caused a logical bug in his code (Issue #43).

- INCOMPREHENSIBLE SUGGESTION refers to the situation where Copilot provides potentially useful code suggestions, but due to the complexity of the code or user's lack of experience, they found it challenging to comprehend the suggested code and need more time to verify its correctness. For example, a user complained that "*My Github Copilot just autocompleted it for me, then I scoured the internet trying to find information pertaining to it but could not*" (SO #73075410).

- SUGGESTION WITH INVALID SYNTAX refers to the situation where the suggestions generated by Copilot may contain syntax errors that prevent the program from running properly. For example, a user found that the code suggestions provided by Copilot "*missed curly brackets, leading to erroneous code when accepting suggestion*" (Discussion #38941).

- LESS EFFICIENT SUGGESTION refers to the code suggestions generated by Copilot that are functionally correct and meet the requirements of users, but may suffer from suboptimal execution efficiency or convoluted logic, potentially impacting the overall quality of the code. For example, when a user requested Copilot to "*find the cup with the most water*" and "*find the maximum amount of water in any cup*", the suggested code performed adequately but was not optimized for efficiency (Issue #160).

- INSECURE SUGGESTION refers to the code suggestions generated by Copilot that introduce security vulnerabilities. For example, a user indicated that the code suggestion lacked accountability for the sizes being read (Discussion #6636).

Interpretation: The quality of code suggestions is a critical factor in determining the capability of Copilot for practical code development. We identified a relatively small number of *Suggestion Content Issues*, possibly indicating that users are less inclined to report problems related to suggested code compared to usage-related problems. Among these problems, LOW QUALITY SUGGESTION (25), NONSENSICAL SUGGESTION (12), and SUGGESTION WITH BUGS (8) are the three most frequently reported types, while INSECURE SUGGESTION (2) and LESS EFFICIENT SUGGESTION (2) are less prevalent. This result implies that the main concern for users could be whether Copilot can provide code suggestions that have significant referential value.

3.1.5. User Experience Issue (4.3%)

This category covers user feedback on their experience of using Copilot. Compared with *Operation Issue*, Copilot

generally runs and functions as intended, but the user experience is suboptimal. *User Experience Issue* can provide insights into areas where Copilot could be improved. *User Experience Issue* can be further classified into four types, which are detailed below.

- POOR FUNCTIONALITY EXPERIENCE refers to a type of user experience problem where the usage of Copilot's code generation-related functionalities is unsatisfactory. When such problems arise, Copilot's functionalities remain operational, which is different from FUNCTIONALITY FAILURE. However, users expressed dissatisfaction with their experience when using these functionalities. These problems can often hinder the coordination between users and Copilot, and even decrease the efficiency of development work. Such problems often highlight areas where Copilot could be further enhanced and may potentially motivate users to propose some *Feature Requests*. For example, a user felt that the code automatically generated and popped up by Copilot was quite noisy (Issue #97).

- POOR SUBSCRIPTION EXPERIENCE refers to the obstacles that users encounter during the process of subscribing to the services of Copilot. Copilot offers several subscription methods (e.g., student verification, paid subscription), leading to some inconvenience for users during this process. For example, one user felt lost and was unsure about what to do next after setting up a billing (Discussion #19119).

- POOR PERFORMANCE refers to performance issues that occur when Copilot is running, which directly impacts the user experience. These problems include high CPU usage, long response times, and overly frequent server access. For example, a user complained that Copilot took "*around 1-2 minutes for it to show one suggestion*" on VSCode, which was very slow (Discussion #19491).

- POOR AUTHENTICATION EXPERIENCE refers to the inconvenience that users encounter when authenticating their identities before using Copilot. While users successfully navigate the login procedure, they may experience a suboptimal user experience. This could be due to factors such as an unwieldy process flow or the absence of explicit instructions. For example, a user complained that Copilot frequently "*prompt to enable GitHub Copilot on every VSCode launch*" which can be a significant source of frustration (SO #70065121).

Interpretation: *User Experience Issues* provide valuable insights into the direction for improving Copilot. Among the POOR FUNCTIONALITY EXPERIENCE (24), the most commonly reported problems involve Copilot's inline suggestions that cause disruptions to the coding process of users (5) and the inconvenience of not being able to accept certain portions of the suggested code (2). These concerns

align with some of the demands mentioned by users in *Feature Request*, e.g., setting when Copilot can generate code and the length of suggested code.

3.1.6. Copyright and Policy Issue (3.3%)

Copilot is trained on a large corpus of open source code and generates code suggestions based on the users' code context. The way in which Copilot operates raises concerns regarding potential copyright and policy issues, as expressed by some users. These problems are divided into three types, as shown below.

- CODE COPYRIGHT ISSUE refers to the concerns raised by some code authors regarding the unauthorized use of their open-source code by Copilot for model training. GitHub is currently one of the most popular web-based code hosting platforms, and since the release of Copilot, there have been suspicions among some code authors that their code hosted on GitHub has been used for training without proper consideration of their license. For example, a user "*started migration*" of his projects on GitHub because he was worried about his code being used to train Copilot without permission (Issue #148).

- CODE TELEMETRY ISSUE refers to the concerns expressed by users regarding Copilot collecting their code to generate suggestions, which may potentially result in the leakage of confidential code. Some users may also simply be unwilling to have their own code, as well as the code generated by Copilot for them, collected for other purposes. For example, a user is concerned that using Copilot might "*give away API key*" in his code (SO #70559637).

- VIOLATION OF MARKETPLACE POLICY is a specific case where a user reported that Copilot was able to be published on the VSCode marketplace despite using proposed APIs, while other plugins were prohibited. The user suspected that this behavior may be in violation of the Marketplace Policy (Issue #3).

Interpretation: The emergence of *Copyright and Policy Issue* reveals the concerns of users about the way Copilot works. Copilot is trained on multi-language open-source code and also needs to collect users' code context during its operation to generate suggestions. These two facts have led people to pay more attention to copyright and intellectual property problems when using Copilot, especially in in-house development.

3.2. Type of Causes (RQ2)
3.2.1. Results

As mentioned in Section 2.4.2, not all problems have corresponding causes that can be extracted. As a result, we identified a total of 391 causes, which were collected from 28.9% of all problems related to Copilot usage, and categorized into 16 types as presented in Table 2. The result indicates that the most frequent causes are *Copilot Internal*

Table 2
Causes of Copilot usage problems

Type	Example (Extracted Cause)	Count	%
Copilot Internal Error (CIE)	*There has been an outage of one of our models which has caused the others to have to handle a higher traffic load.* (Discussion #14370)	76	19.4%
Network Connection Error (NCE)	*It seems that my company's network settings blocked the connection to copilot and that affect the behavior of my personal mac too.* (Discussion #36152)	53	13.6%
Editor/IDE Compatibility Issue (EICI)	*This is related to the version of your visual studio 2022.* (SO #71702171)	50	12.8%
Unsupported Platform (UP)	*··· as some Linux distributions ship with Code OSS or VSCodium which Copilot does not currently support.* (Discussion #8015)	32	8.2%
Improper Configuration/Setting (ICS)	*The settings.json file had disabled inline suggestions.* (SO #76257401)	27	6.9%
Poor Functionality Experience (PFE)	*Please add shortcut command for suggestion ... Feature auto suggestion is something annoying.* (Discussion #7172)	26	6.6%
For Coding Habit (FCH)	*I would like to change the Tab keybinding for completion in vim to something else because many people use the tab binding for opening the completion menu.* (Discussion #6919)	25	6.4%
User Unauthorized (UU)	*I'm sorry that you have not been given access yet.* (Discussion #16795)	22	5.6%
Improper User Operation (IUO)	*I had signed up to Copilot under a different GitHub account* (Discussion #19556)	17	4.3%
Plug-in Compatibility Issue (PCI)	*This was a bad extension interaction between Copilot and learn-markdown.* (Issue #438)	15	3.8%
Intentional Design of Copilot (IDC)	*Yes, this is intentional. There is a hidden setting that you can toggle to disable this behavior.* (Issue #152)	12	3.1%
Unimplemented Feature (UF)	*... currently the Copilot VSCode extension does not support proxies.* (Discussion #11630)	12	3.1%
For Higher Coding Efficiency (FHCE)	*Suggestion: Add mapping generator for classes ... Would be nice to have such feature, that would help with every day mapping.* (Discussion #7870)	9	2.3%
Obsoleted Copilot Version (OCV)	*Your copilot extension is out of date* (Discussion #17463)	7	1.8%
License Restriction (LR)	*Due to the licensing of Copilot, it cannot be used in free or open-source software such as code-server* (Issue #122)	4	1.0%
Code Telemetry Issue (CTI)	*I'd like to disable copilot per workspace, so I can use it for open-source projects but not private/work projects.* (Discussion #47991)	4	1.0%

Error (19.4%) and *Network Connection Error* (13.6%), with *Editor/IDE Compatibility Issue* (12.8%) and *Unsupported Platform* (8.2%) also commonly reported. The example, count, and proportion of each type of cause are presented in Table 2. It is worth noting that certain types of problems can potentially be the causes of other problems. For example, EDITOR/IDE COMPATIBILITY ISSUE is a type of *Compatibility Issue*. However, it is also the causes leading to other types of problems such as INSTALLATION ISSUE and STARTUP ISSUE.

- *Copilot Internal Error* (CIE) refers to the problems with the Copilot server side that affects its usage. These may include errors with the CodeX model and services provided by the server side, which are not visible to users. For example, the Copilot team once reported that "*there has been an outage of one of our models which has caused the others to have to handle a higher traffic load*", which consequently results in temporary issues (Discussion #14370).

- *Network Connection Error* (NCE) refers to the disruptions in the network communication between the user side and the Copilot server side, resulting in an inability for users to utilize the code generation service of Copilot. For example, a user repeatedly encountered issues with Copilot not functioning properly because his "*company's network settings blocked the connection to copilot*" (Discussion #36152).

- *Editor/IDE Compatibility Issue* (EICI) refers to situations where the code editors and IDEs is not compatible with Copilot, resulting in various anomalies when using Copilot. For example, a user was "*unable to install GitHub Copilot extension in Visual Studio 2022 Enterprise*", because the version of his IDE was outdated, resulting in its incompatibility with Copilot (SO #71702171).

- *Unsupported Platform* (UP) refers to situations where users try to use Copilot on development platforms that lack official support for Copilot, which may results in unpredictable problems. For example, the Copilot team claimed that "*Code OSS and VSCodium*" were not supported by Copilot, which explains why some users failed to install Copilot in these two IDEs (Discussion #8015).

- *Improper Configuration/Setting* (ICS) refers to situations where Copilot operates abnormally or offers a suboptimal user experience due to the settings that are not configured appropriately. For example, a user found that "*the setting.json file had disabled inline suggestions*", which resulted in Copilot not providing code suggestions in VSCode (SO #76257401).

- *Poor Functionality Experience* (PFE) refers to the negative experiences users encounter when coding with Copilot, which lead them to request for enhancement to existing features. For example, a user wanted

Copilot to provide code suggestions via a shortcut, because "*feature auto suggestion is something annoying*" (Discussion #7172).

- *For Coding Habit* (FCH) refers to the wishes of some users for Copilot to offer new features and ensure compatibility with their preferred development platforms to accommodate their coding habits. For example, a user asked "*if there was an option for changing the keybinding for accepting the suggestions*", because he was accustomed to "*using the tab binding for opening the completion menu*" (Discussion #6919).

- *User Unauthorized* (UU) refers to situations where some users cannot access Copilot services because they lack the authorization for using Copilot. For example, a user found that "*GitHub Copilot could not connect to server*", because he was not authorized yet (Discussion #16795).

- *Improper User Operation* (IUO) refers to situations where Copilot exhibits unintended behavior because of user mistakes or oversights during the process of using Copilot such as registration, login, and subscription. For example, a user consistently received a prompt in Visual Studio stating that "*Your Copilot experience is not fully configured, please complete your setup*". It was found that this issue arose because he "*had signed up to Copilot with a different GitHub account*" (Discussion #19556).

- *Plug-in Compatibility Issue* (PCI) refers to situations where Copilot fails to work properly because of incompatibility with other plug-ins. For example, "*a bad extension interaction between Copilot and learn-markdown*" resulted in some users being unable to accept code suggestions from Copilot (Issue #438).

- *Intentional Design of Copilot* (IDC) refers to situations where what some users may perceive as anomalies of Copilot are features deliberately designed by the Copilot team. For example, a user noticed that Copilot's "*inline completions are trumping normal completions*". However, a member from the Copilot team proved that "*this is intentional*" and can be disabled by modifying certain setting (Issue #152).

- *Unimplemented Feature* (UF) refers to the functionalities that users assume Copilot to already have, but which the Copilot team has not yet provided. For example, a user could not activate Copilot behind proxy in VSCode, and a Copilot team member explained that "*currently the Copilot VSCode extension does not support proxies*" (Discussion #11630).

- *For Higher Coding Efficiency* (FHCE) is the reason that some users wish for new functionalities in Copilot to enhance efficiency in coding tasks. For example, a user suggested that Copilot could "*add mapping generator for classes*" to facilitate the generation of mapping methods between different classes (Discussion #7870).

- *Obsoleted Copilot Version* (OCV) refers to an outdated version of Copilot that is no longer operational. For example, a user encountering an issue with Copilot not working and was informed that "*your Copilot extension was outdated*" (Discussion #17463).

- *License Restriction* (LR) refers to situations where some development platforms are unable to be integrated with Copilot due to the restriction imposed by Copilot license. For example, a repository contributor explained that Copilot "*cannot be used in free or open-source software such as code-server*" due to its license restriction (Issue #122).

- *Code Telemetry Issue* (CTI) is the reason that some users request new features, aiming to prevent the exposure of their code to Copilot. For example, a user wanted to "*disable Copilot per workspace*", so that he could "*use it for open-source projects but not private/work projects*" (Discussion #47991).

3.2.2. Causes to Problems Mapping

Table 3 illustrates the mapping relationship of Copilot related problems to their causes. We use abbreviations to represent each type of cause; for example, "CIE" represents *Copilot Internal Error*. The full names for all types of causes are provided in the note of Table 3.

Over one third of *Operation Issues* (37.2%) have associated causes. Specifically, AUTHENTICATION FAILURE is primarily induced by CIE, NCE, and UU; FUNCTIONALITY FAILURE is mainly caused by CIE, ICS, and PCI; The occurrence of INSTALLATION ISSUE typically stems from EICI and UP; STARTUP ISSUE commonly originates from CIE and NCE; while ACCESSING FAILURE is primarily attributed to NCE; and VERSION CONTROL ISSUE is mainly brought about by CIE, EICI, and UP.

For *Compatibility Issue*, causes are identified in 9.5% of the cases. To be specific, EDITOR/IDE COMPATIBILITY ISSUE is mainly caused by UP and ICS, while PLUG-IN COMPATIBILITY ISSUE is attributed to ICS and IDC.

For *Feature Request*, four types of causes are identified in 31.0% of cases, which are FCH, PFE, FHCE, and CTI. PFE and FCH are the prime causes for users to raise FUNCTION REQUESTS, while INTEGRATION REQUESTS are mainly attributed to FCH.

For *User Experience Issue*, causes are identified in 19.0% of the cases. However, there are only a few causes identified for each type of *User Experience Issue*, with CIE identified as the prime cause for POOR AUTHENTICATION EXPERIENCE and POOR PERFORMANCE.

For *Suggestion Content Issue*, causes are identified in 8.6% of the cases. CIE and UF are the causes leading to LOW QUALITY SUGGESTION, while CIE and ICS are the causes for NONSENSICAL SUGGESTION.

Table 3
Mapping between problem types (vertical) and cause types (horizontal)

		CIE	NCE	EICI	UP	ICS	PFE	FCH	UU	IUO	IDC	UF	PCI	FHCE	LR	OCV	CTI
Compatibility Issue	Editor/IDE Compatibility Issue	2	0	1	3	3	0	0	0	1	1	0	0	0	0	0	0
	Keyboard Compatibility Issue	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Plug-in Compatibility Issue	2	0	0	0	4	0	0	0	0	3	0	0	0	0	0	0
Copyright and Policy Issue	Code Telemetry Issue	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Code Copyright Issue	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Violation of Marketplace Policy	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Feature Request	Function Request	0	0	0	0	0	21	11	0	0	0	0	0	8	0	0	4
	Integration Request	0	0	0	0	0	0	14	0	0	0	0	0	1	0	0	0
	Professional Copilot Version	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	UI Request	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0
Suggestion Content Issue	Insecure Suggestion	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Less Efficient Suggestion	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Low Quality Suggestion	1	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
	Nonsensical Suggestion	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	Suggestion with Bugs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Incomprehensible Suggestion	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Suggestion with Invalid Syntax	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Operation Issue	Accessing Failure	3	11	0	1	0	0	0	4	0	1	3	0	0	0	0	0
	Authentication Failure	25	16	8	9	2	0	0	13	5	2	0	0	0	1	1	0
	Functionality Failure	17	6	10	0	14	0	0	1	4	6	5	11	0	0	3	0
	Installation Issue	0	0	11	12	0	0	0	0	0	0	0	0	2	0	0	0
	Startup Issue	19	19	15	4	3	0	0	4	5	0	0	3	0	1	1	0
	Version Control Issue	2	0	4	3	0	0	0	0	0	0	0	0	0	0	1	0
User Experience Issue	Poor Authentication Experience	2	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
	Poor Functionality Experience	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	Poor Performance	2	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
	Poor Subscription Experience	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Full names of each cause type: CIE: *Copilot Internal Error*; NCE: *Network Connection Error*; EICI: *Editor/IDE Compatibility Issue*; UP: *Unsupported Platform*; ICS: *Improper Configuration/Setting*; PFE: *Poor Functionality Experience*; FCH: *For Coding Habit*; UU: *User Unauthorized*; IUO: *Improper User Operation*; IDC: *Intentional Design of Copilot*; UF: *Unimplemented Feature*; PCI: *Plug-in Compatibility Issue*; FHCE: *For Higher Coding Efficiency*; LR: *License Restriction*; OCV: *Obsoleted Copilot Version*; CTI: *Code Telemetry Issue*.

Copyright and Policy Issue refers to the concerns of some users about code leakage to Copilot, which inherently forms the reason for raising such problems, thereby no need for further identification of the underlying causes.

3.2.3. Interpretation

Frequency of Causes: CIE is the most common type of cause leading to Copilot usage problems. Typically, the cause identification of CIE relies on user feedback regarding abnormal usage experiences of Copilot, and it often results in a group of users reporting the same problem within a certain time period. For example, "*a bad deployment*" of the Copilot server caused a group of users to report AUTHENTICATION FAILURES (Discussion #39533). The high number of NCE, EICI, and ICS indicates that some problems arise from the environment in which Copilot operates. A common situation of NCE is that users connect to the Copilot server through an HTTP proxy, which may lead to the intercept of Secure Socket Layer (SSL). However, Copilot now offers support for access through an HTTP proxy, thus addressing such problems (GitHub, 2024b). PFE, FCH, FCE, and CTI are the four types of causes for *Feature Request*. The remaining eight types of causes are less common, but can still provide insights into specific problems related to Copilot. For instance, UU is identified as the direct cause for many users experiencing AUTHENTICATION FAILURE and FUNCTIONALITY FAILURE when using Copilot.

Mapping of Causes to Problems: When Copilot users encounter *Operation Issues*, nearly one quarter (23.1%) of these problems are caused by errors originating from Copilot server (i.e. CIE), while the rest of such problems are induced by the environment on which Copilot operates. According to

the causes of *Feature Request*, it appears that users typically request for new functionalities or enhancement of existing features based on their personal coding habits and suboptimal experiences when coding with Copilot. Fewer causes have been identified for *Compatibility Issue*. When users identify the problem as *Compatibility Issue*, they tend to focus on finding solutions rather than further analyzing the causes. The causes identified for *Suggestion Content Issue* and *Copyright and Policy Issue* are limited in number. One possible reason is that the non-open source nature of Copilot prevents users from investigating the causes of the problems in these two categories. From the results of Table 3, we did not identify the main causes leading to *User Experience Issue*. However, when users encounter POOR AUTHENTICATION EXPERIENCE or POOR PERFORMANCE when using Copilot, consideration should be given to whether there are internal errors within Copilot, as the four cases of CIE leading to such problems.

3.3. Type of Solutions (RQ3)
3.3.1. Results

As mentioned in Section 2.4.2, not all problems have corresponding solutions that can be extracted. As a result, we identified a total of 497 solutions, which were used to address 36.7% of all problems, and categorized into 11 types as shown in Table 4. The result reveals that most of the usage bugs were addressed by *Bug fixed by Copilot* (27.2%). When users tried to solve problems themselves, *Modify Configuration/Setting* (22.1%), *Use Suitable Version* (17.1%), and *Reinstall/Restart/Reauthorize Copilot* (12.1%) were commonly used as effective solutions. The example,

count, and proportion of each type of solution are presented in Table 4.

- *Bug Fixed by Copilot* (BFC) refers to the process where the Copilot team collects problems reported by users or conducts functionality test on Copilot, and then addresses identified bugs. For example, a user failed to use Copilot because of the "*problem from Copilot's server-side*", and this problem was fixed by the Copilot team as soon as they noticed it (Issue #86). Typically, the resolution process to the problems by the Copilot team is unclear for users.

- *Modify Configuration/Setting* (MCS) refers to the solutions that users adjust settings or configuration files of Copilot to solve specific usage problems. For example, by "*updating the file setting.json*", a user resolved the problem of Copilot "*not adding closing brackets*" (SO #68347605).

- *Use Suitable Version* (USV) refers to users choosing a suitable version of Copilot that resolves the problems faced in the Copilot version currently used. This type of solution also includes adjusting the version of IDEs or code editors to be compatible with Copilot. For example, a user who wanted to use Copilot in Visual Studio solved the problem of "*not being able to find the GitHub copilot extension*", by "*upgrading Visual Studio to 17.2*" (Discussion #18566).

- *Reinstall/Restart/Reauthorize Copilot* (RC) is a type of solution allowing users to reset Copilot to its original state, thereby resolving any previous errors or restoring settings to their default conditions. For example, a user found that "*GitHub Copilot could not connect to server*" in VSCode, and solved it by simply "*restarting extension*" (Discussion #27378).

- *Feature Implemented by Copilot* (FIC) refers to the introduction or enhancement of a feature by the Copilot team. For example, the latest Copilot "*contains preliminary support for connecting through HTTP proxy servers*", addressing the AUTHENTICATION FAILURE previously experienced by some users (SO #73242748).

- *Follow Official Instruction* (FOI) refers to the process where users follow the steps provided by the Copilot UI or described in the user manuals during registration, login, subscription, and configuration. This type of solution usually aimed at assisting users who are not familiar with Copilot. For example, a user was advised to consult the "*Copilot doc*" for guidance on configuring keyboard shortcuts on Mac (SO #75276040).

- *Disable Interfering Factors* (DIF) refers to the process where users identify and remove the factors disrupting the normal operation of Copilot, such as other activated plug-ins and HTTP proxies that could interfere with network connection. For example, a user reported

that "*GitHub Copilot autosuggesions are not auto-filled in .md files*", and solved this problem by disabling the "*Markdown all in one*" plug-in (Discussion #10203).

- *Restart Runtime Environment* (RRE) refers to the process of restarting the environment Copilot operates on to resolve any errors affecting the current runtime environment. For example, restarting the computer and IDE on which Copilot is running helped a user solve an INSTALLATION FAILURE on IntelliJ IDEA 2022 (Discussion #17638).

- *Modify the Input Way* (MIW) refers to users changing the way they prompt Copilot to generate code, to address problems that Copilot does not automatically provide code suggestions, or to make Copilot provide codes that better meet their expectations. For example, "*going to a new line after a code comment, and starting typing what you want to create*" resolved the problem that Copilot created "*the same comment over and over again*" (SO #70995718).

- *Install/Update Framework* (IUF) refers to installing or updating the frameworks in IDEs and code editors for Copilot to operate. This solution is primarily employed to resolve the incompatibility between Copilot.vim (Pope, 2024) and Node.js. For example, a user found that he had to "*install node.js*" for Copilot to work in Neovim (Discussion #40300).

- *Others*: Different from Copilot usage problems and their causes, which can be categorized into specific categories and types, several dedicated solutions aimed at addressing certain Copilot usage problems are classified under the category *Others*. For example, using the VSCode extension "*Win-CA*" in append mode to resolve ACCESSING FAILURE of Copilot was identified only once, making it difficult to form a new type of solution (SO #71367058).

3.3.2. Solutions to Problems Mapping

Table 5 illustrates the mapping relationship of Copilot related problems to their solution types, using abbreviations to represent each type of solution. For example, "BFC" represents *Bug Fixed by Copilot*. The full names for all types of solutions are provided in the note of Table 5.

For *Operation Issue*, 44.5% of the cases are effectively addressed. Specifically, ACCESSING FAILURE is mainly addressed by BFC; AUTHENTICATION FAILURE is primarily resolved by BFC, MCS, and RC; FUNCTIONALITY FAILURE is commonly fixed by BFC, MCS, and USV; and STARTUP ISSUE is mostly solved by BFC, MCS, USV, and RC. To address INSTALLATION ISSUE and VERSION CONTROL ISSUE, USV is the primarily employed solution.

For *Compatibility Issue*, effective solutions were identified in 39.3% of the cases. BFC, MCS, and USV are the

Table 4
Solutions of Copilot usage problems

Type	Example (Extracted Solution)	Count	%
Bug Fixed by Copilot (BFC)	*The problem is from Copilot's server-side and got fixed now.* (Issue #86)	135	27.2%
Modify Configuration/Setting (MCS)	*Update your settings.json like the following...* (SO #68347605)	110	22.1%
Use Suitable Version (USV)	*Upgrading to 17.2 solved the problem.* (Discussion #18566)	85	17.1%
Reinstall/Restart/Reauthorize Copilot (RC)	*Problem solved after restarting extension.* (Discussion #27378)	60	12.1%
Feature Implemented by Copilot (FIC)	*The latest release of Copilot for Visual Studio Code and Copilot for IntelliJ contains preliminary support for connecting through HTTP proxy servers.* (SO #73242748)	32	6.4%
Follow Official Instruction (FOI)	*In the Copilot docs, see for Visual Studio Code "Keyboard shortcuts for GitHub Copilot"* (SO #75276040)	15	3.0%
Disable Interfering Factors (DIF)	*Disable Markdown All in One, and it will work.* (Discussion #10203)	15	3.0%
Restart Runtime Environment (RRE)	*Restart your machine and IDE as a quick fix.* (Discussion #17638)	11	2.2%
Modify the Input Way (MIW)	*Go on new line after comment, and start typing what you want to create.* (SO # 70995718)	8	1.6%
Install/Update Framework (IUF)	*Have to install node.js for it to work.* (Discussion #40300)	5	1.0%
Others	*Installing Win-CA helps hides it again and all looks back to normal.* (SO #71367058)	21	4.2%

commonly used solutions for resolving EDITOR/IDE COMPATIBILITY ISSUE, while BFC, MCS, and USV are mostly employed for addressing PLUG-IN COMPATIBILITY ISSUE. Moreover, resolving KEYBOARD COMPATIBILITY ISSUE mainly relies on MCS.

For *Feature Request*, 21.7% of the cases have effective solutions. Specifically, FUNCTION REQUEST is primarily addressed by MCS and FIC. Two cases of UI REQUEST were addressed by MCS. In addition, although the solutions for PROFESSIONAL COPILOT VERSION have not been identified, GitHub announced the availability of Copilot Enterprise on February 27, 2024 (GitHub, 2024a), potentially addressing the need for a professional version of Copilot.

We identified effective solutions for 24.1% of *User Experience Issues*. POOR AUTHENTICATION EXPERIENCE is addressed by BFC, USV, and RC, while POOR FUNCTIONALITY EXPERIENCE is mainly resolved by MCS. Moreover, BFC is the only solution identified for addressing POOR PERFORMANCE.

We only identified five solutions for *Suggestion Content Issue*, accounting for 10.2% of the total cases. BFC and MIW were employed to address NONSENSICAL SUGGESTION and LOW QUALITY SUGGESTION ISSUES, and no solutions have been identified for resolving INSECURE SUGGESTION, LESS EFFICIENT SUGGESTION, SUGGESTION WITH BUGS, INCOMPREHENSIBLE SUGGESTION, and SUGGESTION WITH INVALID SYNTAX.

Only 6.7% of *Copyright and Policy Issues* have corresponding solutions. For CODE COPYRIGHT ISSUE and CODE TELEMETRY ISSUE, MCS is the identified solution.

3.3.3. Interpretation

Frequency of Solutions: BFC is the most commonly employed solution for addressing problems related to Copilot usage, which is reasonable since *Copilot Internal Error* is identified as the most common cause of Copilot usage problems, indicating that many problems of Copilot cannot be resolved by users. MCS, USV, and RC are frequently employed solutions when users attempt to resolve Copilot

related problems by themselves. Typically, users can obtain relevant experience and knowledge in resolving Copilot usage problems by these three methods on public Q&A platforms (e.g., GitHub Issues, GitHub Discussions, SO). FIC ranks as the fifth most frequently used solution, reflecting the expansion and improvement of Copilot features to match the requirements of a large user community. The remaining six types of solution account for only 11.8% of the total number of solutions, but can still provide valuable experiences for users who face specific Copilot related problems. For instance, DIF can effectively resolve the conflicts between the plug-in "Markdown All in One" and "Copilot", saving lots of time for the users who experience the same problem.

Mapping of Solutions to Problems: For *Operation Issue* and *Compatibility Issue*, a great number of effective solutions have been identified. This is partially due to the high number of *Operation Issue* and *Compatibility Issue*, and their direct impact on the proper functioning of Copilot. As a result, both the Copilot team and users are inclined to promptly address these two categories of Copilot related problems. Besides the errors in Copilot server that can be addressed by BFC, some *Operation Issues* and *Compatibility Issues* are attributed to the Copilot running environments of users. Therefore, users can address the problems in these two categories through MCS, USV, and RC. The relatively limited number of solutions for *Suggestion Content Issue* reflects the lack of effective methods for users to adjust the code suggested by Copilot. *Feature Request* and *User Experience Issue* usually require a new release of Copilot by the Copilot team to expand and optimize the features of Copilot, making it difficult to meet the expectations of some users in a short term. Additionally, we found that MCS is frequently utilized by users when trying to get the functionality they want without waiting the new release of Copilot. Fewer solutions were identified for *Copyright and Policy Issue*, indicating that it is important to provide satisfactory solutions to address the concerns of code leakage.

Table 5
Mapping between problem types (vertical) and solution types (horizontal)

		BFC	MCS	USV	RC	FIC	FOI	DIF	RRE	MIW	IUF	Others
Compatibility Issue	Editor/IDE Compatibility Issue	18	10	12	0	1	1	1	0	1	0	0
	Keyboard Compatibility Issue	0	2	0	0	0	0	0	0	0	0	0
	Plug-in Compatibility Issue	14	16	1	0	0	0	3	0	0	0	4
Copyright and Policy Issue	Code Telemetry Issue	0	2	0	0	0	0	0	0	0	0	0
	Code Copyright Issue	0	1	0	0	0	0	0	0	0	0	0
	Violation of Marketplace Policy	0	0	0	0	0	0	0	0	0	0	0
Feature Request	Function Request	0	9	3	0	14	1	0	0	1	0	0
	Integration Request	0	0	0	0	13	0	0	0	0	0	0
	Professional Copilot Version	0	0	0	0	0	0	0	0	0	0	0
	UI Request	0	2	0	0	0	0	0	0	0	0	0
Suggestion Content Issue	Insecure Suggestion	0	0	0	0	0	0	0	0	0	0	0
	Less Efficient Suggestion	0	0	0	0	0	0	0	0	0	0	0
	Low Quality Suggestion	0	0	0	0	0	0	0	0	1	0	2
	Nonsensical Suggestion	2	0	0	0	0	0	0	0	1	0	0
	Suggestion with Bugs	0	0	0	0	0	0	0	0	0	0	0
	Incomprehensible Suggestion	0	0	0	0	0	0	0	0	0	0	0
	Suggestion with Invalid Syntax	0	0	0	0	0	0	0	0	0	0	0
Operation Issue	Accessing Failure	8	4	3	4	2	1	0	2	0	0	2
	Authentication Failure	31	15	7	30	2	1	3	1	0	2	3
	Functionality Failure	27	31	14	8	0	8	5	2	3	0	5
	Installation Issue	2	3	11	1	0	0	0	1	0	0	0
	Startup Issue	23	11	24	15	0	2	3	4	0	3	5
	Version Control Issue	4	1	8	1	0	0	0	1	0	0	0
User Experience Issue	Poor Authentication Experience	2	0	1	1	0	0	0	0	0	0	0
	Poor Functionality Experience	0	3	1	0	0	0	0	0	1	0	0
	Poor Performance	3	0	0	0	0	0	0	0	0	0	0
	Poor Subscription Experience	1	0	0	0	0	1	0	0	0	0	0

Full names of each solution type: BFC: *Bug Fixed by Copilot*; MCS: *Modify Configuration/Setting*; USV: *Use Suitable Version*; RC: *Reinstall/Restart/Reauthorize Copilot*; FIC: *Feature Implemented by Copilot*; FOI: *Follow Official Instruction*; DIF: *Disable Interfering Factors*; RRE: *Restart Runtime Environment*; MIW: *Modify the Input Way*; IUF: *Install/Update Framework*.

4. Implications

4.1. Implications for the Copilot Users

> **Implication 1**. Seeking inspiration from Copilot's code suggestions rather than relying on them can lead to better control over your coding tasks.

In *Feature Request*, we have observed that users frequently present their wish to accept code suggestions word-by-word or line-by-line when reviewing them. As a user complained in an SO post, "*Copilot inputs tons of code when people just want a tiny part*" (SO #74091857). Users' desire for such requests may reflect their concerns about losing control over their coding tasks because Copilot sometimes is too "productive" for them. In the experiment conducted by Vaithilingam et al. (2022), five participants perceived the same worry when they failed to understand the code suggested by Copilot. Thus, finding a right balance between leveraging Copilot's productivity and maintaining developers' command over their code is important for Copilot users. As pointed out by Ziegler et al. (2024), the value of Copilot lies in whether the suggestions serve as a useful starting point for further development. Therefore, a useful position of Copilot would be a tool that provides coding insights, rather than a complete replacement for developers in coding tasks. Before using Copilot for coding, users should ideally have a clear understanding of their code structure. Then, by guiding Copilot to provide code suggestions, they can further optimize the code implementation. This way ensures that

users maintain control over their code while incorporating the inspiring suggestions provided by Copilot, keeping the role of users as a real "pilot" of the coding tasks.

> **Implication 2**. Review the code suggestions provided by Copilot before accepting them to prevent introducing quality issues into your project.

According to the results of RQ1, the emergence of IN-SECURE SUGGESTION, LESS EFFICIENT SUGGESTION, and SUGGESTION WITH INVALID SYNTAX indicates potential quality issues in the code suggested by Copilot. The code quality of Copilot generated code has been a critical topic for the Copilot team and many researchers (e.g., Yetistiren et al. (2022), Siddiq et al. (2022)). Since the outputs of LLMs are unpredictable and the code snippets used to train Copilot potentially contain quality issues, it is unrealistic to expect Copilot to consistently produce quality-assured code under all circumstances. In our study, we identified only a few issues related to the quality of Copilot-generated code. One possible reason is that developers may not be attentive enough and may lack the experience to identify quality issues such as code security in Copilot generated code. According to research conducted by Fu et al. (2024), 29.8% of the 452 code snippets produced by Copilot and collected from GitHub contain security issues spanning 38 Common Weakness Enumeration (CWE) (MITRE, 2023) categories. These code snippets were produced in practical software development rather than controlled environments,

Table 6
Mapping between problem categories (vertical) and IDEs/editors (horizontal)

	VSCode	Neovim	Visual Studio	IDEA	PyCharm	VSCodium	Emacs	PhpStorm	Vim	Others
Compatibility Issue	130	24	16	5	3	2	1	2	2	10
Copyright and Policy Issue	6	0	1	0	0	0	1	0	0	1
Feature Request	63	20	14	7	3	0	5	2	5	14
Suggestion Content Issue	9	2	5	0	1	0	0	0	0	0
Operation Issue	365	63	59	40	36	14	6	11	10	41
User Experience Issue	27	1	0	4	2	0	1	3	0	2

which reveals the severe security challenges of the code produced by Copilot in real-world scenarios. Users need to have the necessary security awareness and skills when dealing with the code suggestions generated by Copilot. Additionally, other quality issues such as code smells (Zhang et al., 2024) and code efficiency (Nguyen and Nadi, 2022) present in Copilot code suggestions also require users to exercise critical evaluation and judgment. We noticed that users may find it challenging to fully understand the code suggestions provided by Copilot in INCOMPREHENSIBLE SUGGESTION, leading to difficulties in recognizing the quality issues introduced by Copilot. Therefore, it is essential to carefully examine whether there are any quality issues with the code suggestions from Copilot before accepting them. If users want to utilize code suggestions from Copilot in critical software projects, rigorous code reviews and thorough testing are necessary.

> **Implication 3**. It is advisable to utilize IDEs or code editors that are officially supported by Copilot.

Table 6 presents the correlation between the number of Copilot usage problems and the IDEs or code editors that users predominantly utilize. We found that discussions regarding Copilot usage problems mainly occurred on mainstream coding development platforms such as VSCode, Neovim, Visual Studio, IDEA, PyCharm. However, the results in Table 6 do not imply that using a particular IDE or code editor will result in encountering a large number of specific category of Copilot-related problems. Higher numbers often indicate that such problems have been widely discussed and even already addressed. For instance, users of VSCode reported many *Operational Issues* that have been resolved through discussions. This does not necessarily mean that using Copilot on VSCode will lead to encountering a significant number of *Operation Issues*. Besides, the Copilot team also actively provides solutions to the queries arising on the platforms officially supported by Copilot. As a result, users utilizing Copilot on these IDEs or code editors are more likely to resolve usage problems by finding existing solutions or reporting them on public Q&A platforms. On the contrary, we found that *Unsupported Platform* ranks as the fourth most frequent cause leading to Copilot usage problems. Based on the results of RQ2 (see Table 3), it is also one of the primary reasons causing various types of *Operation Issue*, which suggests that users may encounter unforeseen problems when attempting to use Copilot on the platforms that are not supported yet. These

problems are often challenging to resolve. In our study, none of the 31 issues associated with *Unsupported Platform* has been resolved effectively. Therefore, users should prioritize using Copilot on officially supported platforms to ensure an optimal experience.

4.2. Implications for the Copilot Team

> **Implication 4**. Provide more customization options to allow users to tailor the behavior of Copilot to align with their own workflow.

Among the 114 FUNCTION REQUESTS, we identified 52 instances of such requests to customize the behavior of Copilot in various aspects, accounting for approximately 50%. Some common requests are *specifying the file types or workspace in which Copilot automatically runs* (11), *modifying the shortcut keys for accepting suggestions* (10), *accepting code suggestions line-by-line or word-by-word* (9), *preventing Copilot from generating certain types of suggestions (e.g., file paths, comments)* (3), and *configuring text color and fonts* (3). Our previous study on the expected feature of Copilot also indicates that the functionalities provided by Copilot have not yet fully met the requirements of users for flexible utilization of the tool (Zhang et al., 2023). Additionally, according to some of the identified POOR FUNCTIONALITY EXPERIENCE (e.g., perceiving the auto-suggestions of Copilot as distracting, which is also mentioned in the study of Bird et al. (2023)), we can discern the demand for customizing the behavior of Copilot. It is believed that the extent to which the behavior of Copilot can adapt well to user coding habits is a vital factor in their decision to use Copilot. Therefore, providing flexible and user-friendly customization options is highly beneficial.

> **Implication 5**. Provide more ways to control the content generated by Copilot.

According to Table 5, there is only a small number of solutions for *Suggestion Content Issue*. Out of the 59 *Suggestion Content Issues*, only five solutions were identified, indicating that users may find it challenging to provide ideal solutions for the problems of content suggested by Copilot. The output of Copilot is inherently unpredictable, and users have limited ways to control the code generated by Copilot besides modifying code context or code comments per se. Based on our observation, features such as *allowing users to define code styles and conventions*, and *choosing multiple*

Table 7
Mapping between problem categories (vertical) and programming languages (horizontal)

	Python	C#	JavaScript	Php	Java	TypeScript	C++	Go	Others
Compatibility Issue	13	8	8	2	1	5	3	1	1
Copyright and Policy Issue	0	1	1	0	0	0	0	0	0
Feature Request	8	8	3	2	1	0	2	1	4
Suggestion Content Issue	10	9	5	1	1	4	2	1	4
Operation Issue	49	13	14	12	15	10	5	4	7
User Experience Issue	5	0	1	3	0	1	1	0	0

files as the context for Copilot to generate code suggestions are worth trying.

> **Implication 6**. Simplify the configuration of Copilot and provide support for more IDEs and code editors.

According to the results of RQ1 and RQ2, *Compatibility Issue* is the second-largest category, and *Editor/IDE Compatibility Issue* is one of the main causes that leads to many *Operation Issues*. From the perspective of users, we also have observed lots of discussions related to configuration and settings of Copilot, which makes *Modify Configuration/Setting* the second most frequently employed solution. Additionally, *Improper Configuration/Setting* is the fifth most common cause of problems related to Copilot usage. Based on these findings, we believe that simplifying the configuration process of Copilot for users can significantly improve their experience. For example, the Copilot team may offer more detailed installation and configuration guidelines, provide user-friendly configuration options. Moreover, we have identified 71 INTEGRATION REQUESTS, indicating a significant number of users expressing a desire for Copilot compatibility across a broader range of IDEs and code editors.

> **Implication 7**. Increase the diversity of the code suggested by Copilot while improving their quality.

In *Suggestion Content Issues*, the predominant types are LOW QUALITY SUGGESTION and NONSENSICAL SUGGESTION. Bird et al. (2023) also observed that Copilot occasionally offers peculiar and nonsensical code suggestions as reported by the Copilot users. Other prior studies (e.g., Vaithilingam et al. (2022), Pearce et al. (2022)) also indicated the problems with the code suggestions provided by Copilot in terms of code quality. However, compared to the issues of specific code content suggested by Copilot, we found that users are more inclined to discuss situations where Copilot fails to provide valuable code suggestions as problems on public Q&A platforms. This tendency actually reflects one of the primary purposes for which many users utilize Copilot: they hope that Copilot will inspire their ideas in coding. Observations from some instances of *User Experience Issue* have shown how users can be disappointed when all 10 code suggestions provided by Copilot lack diversity. In fact, in addition to the improvement in code quality, diverse coding suggestions are essential for satisfying the needs of users for more inspirational code.

> **Implication 8**. When using popular programming languages, user experience with Copilot shows no noticeable differences. However, additional enhancement may be necessary for more niche or domain-specific languages.

Table 7 presents the correlation between the number of Copilot usage problems and the programming languages that users predominantly utilize. Python, C#, JavaScript, and Java are the most frequently used programming languages. The official claim from the Copilot team is that Copilot can support almost all programming languages (GitHub, 2024d). Based on our observations of these reported problems, no commonly used programming language consistently exhibited specific categories of problems for users. Users of these mainstream languages have not indicated significant differences in user experience and the quality of code suggestions when using different languages. However, We found that some users expressed a need for enhancements or support for niche and domain-specific languages when using Copilot. For instance, a user expressed disappointment with the code suggested by Copilot for the Nim language (Discussion #7180), while another user requested support for Classic ASP and VBScript (Discussion #53154). Due to varying training data across different programming languages, users may have a suboptimal experience when using less common languages. Therefore, we suggest that the Copilot team collect more relevant training data and further train Copilot for these programming languages to improve the user experience.

> **Implication 9**. Consider intellectual property and copyright when gathering training data for Copilot and giving code suggestions.

Since the announcement of Copilot, there has been ongoing controversy regarding its implications for intellectual property and copyright. In our dataset, the number of *Copyright and Policy Issue* is higher than we expected, and we observed many related concerns from both users and repository owners during the data extraction process. For the owners of the repositories hosted on GitHub, the primary concern is whether their code has been or will be used illegally to train Copilot. Copilot users are concerned about whether the sensitive code context provided for generating code suggestions would be collected and stored by Copilot. In *Feature Request*, we noticed that some users call for Copilot to introduce new features that prevent CODE TELEMETRY ISSUE. In addition, Bird et al. (2023)

found some discussions between Copilot users about how copyright applied to code suggestions generated by Copilot. According to their findings, Copilot users hold different views about how to license the code generated by Copilot and whether the code generated by Copilot should be capable of copyright protection. Issues related to intellectual property and copyright when using Copilot should be seriously taken into account by the Copilot team to prevent potential impacts on the development of Copilot and user experience. As Akbar et al. (2023) pointed out, intellectual property, legal compliance, and privacy are critical ethical principles when developing and using LLM-driven tools like Copilot. Therefore, we contend that the Copilot team should take measures to address these problems, providing high-quality code generation services while protecting user privacy and intellectual property. Besides, we noticed that the Copilot team has published "GitHub Copilot Trust Center" (GitHub, 2024c) to help answer questions regarding privacy, security, and intellectual property. We plan to further explore the role of this center in addressing user concerns regarding the *Copyright and Policy Issue* in our future work.

4.3. Implications for Researchers

> **Implication 10**. The use of Copilot may alter the coding process and increase the time cost of verifying code suggestions, making code explanation highly important.

In our research, INCOMPREHENSIBLE SUGGESTION ranks as the fourth most common *Suggestion Content Issue*, and the code explanation feature is also frequently requested. Some users mentioned that the code suggestions generated by Copilot are excessively long, resulting in reduced readability. This indicates that when Copilot provides relatively complex suggestions, or when users lack coding experience in a particular domain, understanding the code logic and verifying its correctness can be time-consuming. In that case, users may refrain from adopting code suggestions of Copilot as it makes them feel a loss of control over the coding task, according to the research of Vaithilingam et al. (2022). The study by Wang et al. (2023b) also shows that using AI-generated code can lead to substantial review pressure. Therefore, we believe that AI coding tools like Copilot will change the allocation of time spent on various tasks in software development. As expected by some users, the code explanation feature would be helpful. GitHub Copilot Chat now is generally available for organizations and individuals (Zhao, 2023), which is powered by GPT-4. The chat feature enables Copilot to interact more flexibly with users, providing the potential for implementing functions such as code explanation, code refactoring, and vulnerability detection. For future research, we plan to explore how the chat feature can influence the programming process and enhance efficiency of coding tasks.

> **Implication 11**. To accurately gauge user satisfaction with Copilot, it is essential to consider programming tasks across various application domains, as well as the purposes for which users employ Copilot.

We identified relatively few *Suggestion Content Issues*, and one potential reason is that users are less inclined to report specific code-related issues to public Q&A platforms, indicating a certain degree of tolerance for AI-generated erroneous suggestions (Weisz et al., 2021). Therefore, our study results may not fully reflect user satisfaction with the Copilot generated code content. Users often propose new *Feature Requests* based on their individual needs and coding habits, coupled with their experience of using Copilot. Some of these requests are closely associated with the domains of the applications that Copilot users develop, such as front-end web development and game development. Consequently, user satisfaction levels with Copilot may vary across different application domains. Thus, assessing the capabilities of Copilot through the lens of single programming tasks, such as addressing algorithmic problems or writing SQL statements, may not represent the actual user satisfaction in other application domains. Furthermore, our previous research found that users have diverse objectives when using Copilot (Zhang et al., 2023), directly influencing their expectations. For instance, users looking to quickly grasp new technologies with the assistance of Copilot are more concerned with whether the code suggestions provide helpful programming guidance for subsequent steps; conversely, those who wish to use Copilot for repetitive coding tasks, such as CRUD operations on a database, are more concerned with whether Copilot can generate correct code to reduce the significant effort they spend on fixing similar issues in code suggestions. Therefore, the intended utilization of Copilot also stands as a critical factor influencing the satisfaction of users with Copilot. A focused study on user satisfaction with Copilot based on these two factors will provide an insightful assessment.

> **Implication 12**. By analyzing the timeline of when problems related to Copilot usage occurred, we found that *Feature Requests* of users increased with the iterations of Copilot.

Fig. 4 illustrates the variation in the number of six categories of Copilot usage problems every three months. As shown in Fig. 4, *Operation Issue* outnumbers other categories during almost all the periods, which is reasonable since *Operation Issues* constitute 57.5% of all Copilot-related problems. Besides, we found that the numbers of *User Experience Issue*, *Suggestion Content Issue*, and *Copy and Policy Issue* have remained relatively stable. However, despite the Copilot team introducing new features and enhancing existing features to meet the requirements of Copilot users, the number of *Feature Requests* has continued to rise since January 2022. Considering the uncertainty of the code generated by Copilot, users are also uncertain about the
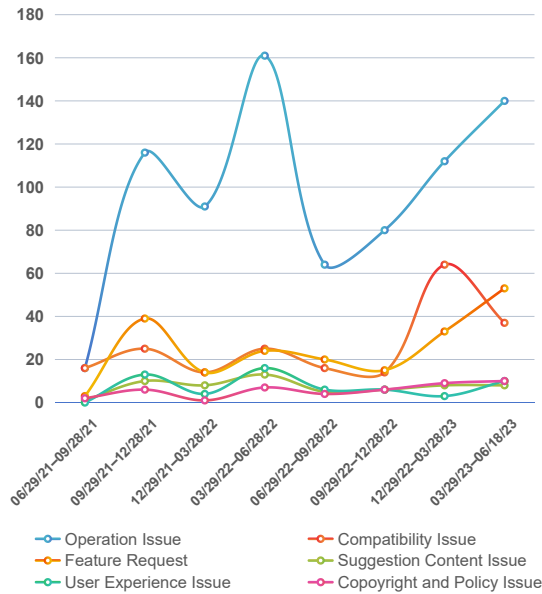
Figure 4: Trends of Copilot Problems

optimal way to interact with Copilot. Consequently, users provide feedback based on their practical coding experience with Copilot and hope that the optimizations from the Copilot team will enhance their usage experience. Even though the Copilot team continuously introduced new functions to meet user expectations, users can still provide more feedback and come up with additional demands based on their experience with these new features of Copilot. As the version of Copilot evolves, user expectations also evolve correspondingly. Hence, exploring how AI code generation tools such as Copilot can better interact with users could be an interesting and worthy research topic. We argue that researchers should consistently monitor the feedback and trends in *Feature Requests* for Copilot, particularly focusing on the feedback following the introduction of new features, which can guide future enhancements and pinpoint specific areas where Copilot can be better improved.

5. Threats to Validity

The threats to validity are discussed according to the guidelines in Runeson and Höst (2009), and internal validity is not considered, since we did not investigate the relationships between variables and results.

Construct validity: As the processes of data labelling, data extraction, and data analysis in this study were conducted manually, there is a risk of introducing personal bias. Therefore, we implemented some strategies to enhance the construct validity. In order to reduce this threat, the first and third authors carried out pilot experiments to agree on the criteria for data labelling and data extraction. If any disagreements arose during these processes, the second

author was involved to achieve a consensus. The results of data extraction were rechecked by the three authors to ensure accuracy. The data analysis was conducted by the first author. When uncertainties emerged, the first author discussed them with the second and third authors to achieve joint agreement. For the results of the data analysis, the negotiated agreement approach (Campbell et al., 2013) was employed to address any conflicts.

External validity: For our research, the primary threat to external validity is the selection of data sources. To maximize external validity, we chose GitHub Issues, GitHub Discussions, and SO posts as data sources. GitHub Issues is a tool used to report and track software issues, allowing users to report errors, request features, and raise questions to developers. While GitHub Discussions is a new feature on GitHub that aims to provide a more open and organized platform for users to communicate and share insights with other community members. As a popular Q&A community, Stack Overflow is also a platform for many developers to engage in discussions and share insights regarding Copilot usage. These platforms contain a substantial amount of relevant data, and their data are complementary to each other. Consequently, we were able to collect diverse usage-related data of Copilot from a large number of developers and projects from these three data sources. However, despite all these efforts, we admitted that there may still be relevant data that we missed. We collected the discussions under the "Copilot" subcategory of GitHub Community rather than searching for all GitHub discussions by using "Copilot" as the keyword. The absence of discussions from other repositories that are potentially related to Copilot could affect the statistics on the number of different categories of Copilot usage problems. Additionally, with the rapid release of the new versions of Copilot, we acknowledge that some new problems related to Copilot usage may emerge, or the problems reported may have been resolved. Hence, there is a need to continuously collect new data in future work to refine our conclusions. Also, industrial surveys can be employed to obtain more information related to the usage of Copilot, such as development contexts and users' coding experience. To validate and replicate our findings and potentially uncover additional insights, we provide our dataset for other researchers (Zhou et al., 2024).

Reliability: To minimize potential uncertainties arising from the research methodology, we have implemented multiple measures to maximize the reliability of our study. We conducted a pilot labelling to assess the consistency of the two authors prior to the formal data labelling process. The Cohen's Kappa coefficients of the three pilot labelling processes are 0.824, 0.834, and 0.806, indicating good agreement between the authors. Throughout the data labelling, extraction, and analysis process, we thoroughly discussed and resolved any inconsistencies within the team to ensure the consistency and accuracy of the result. Furthermore, we have made available the dataset of the study (Zhou et al., 2024) to enable other researchers to validate our findings.

6. Related Work

6.1. Evaluating the Quality of Code Generated by Copilot

Several studies focused on various aspects of code quality generated by Copilot. Siddiq et al. (2022) analyzed the prevalence of code smells in the datasets used to train code generation tools like Copilot, and observed the presence of 18 types of code smells in the suggestions provided by Copilot. Yetistiren et al. (2022) evaluate the validity, correctness, and efficiency of the Copilot generated code, and their results indicate that Copilot is capable of generating valid code with a success rate of 91.5%. Pearce et al. (2022) prompted Copilot to generate code related to high-risk network security vulnerabilities in order to investigate the conditions that may lead Copilot to suggest insecure code. They found that vulnerabilities were present in 40% of the cases. Nguyen and Nadi (2022) conducted a study using 33 LeetCode problems to evaluate the correctness and comprehensibility of Copilot in four different programming languages. They found that Copilot's suggestions had low cyclomatic and cognitive complexity and did not show significant differences across programming languages. Moradi Dakhel et al. (2023) investigated the capabilities of Copilot in two different programming tasks and found that it was able to provide solutions for almost all basic algorithmic problems, but some of the solutions were buggy and not replicable. The study conducted by Asare et al. (2023) suggests that Copilot won't generate code with the same vulnerabilities as those previously introduced by human developers overall. Sobania et al. (2022) evaluated the performance of Copilot on standard program synthesis benchmark problems and compared it with results from the genetic programming literature. They found that Copilot demonstrated more mature performance. Mastropaolo et al. (2023) aimed to investigate whether different but semantically equivalent natural language descriptions would lead to the generation of the same recommended functions. Their results indicate that differences between semantically equivalent descriptions could affect the correctness of the generated code. Al Madi (2023) investigated the readability and visual inspection of the code generated by Copilot. They revealed the importance of developers being cautious and vigilant when working with code generation tools such as Copilot. Gustavo et al. (2023) investigated the impact of programming with LLMs that support Copilot. They found that users assisted by LLMs produce critical security bugs at a rate no greater than 10% more than those not assisted.

6.2. Copilot's Impact on Practical Development

Several studies focused on investigating the performance of Copilot in actual software development, as well as the opinions of software practitioners on it. Wang et al. (2023a) conducted an interview with 15 practitioners and then surveyed 599 practitioners from 18 IT companies regarding their expectations of code completion. They found that 13% of the participants had used Copilot as their code completion tool. Jaworski and Piotrkowski (2023) prepared a survey questionnaire consisting of 18 questions to investigate developers' attitudes toward Copilot. The research findings indicate that most people have a positive attitude towards the tool, but few participants showed concerns about security issues associated with using Copilot. Imai (2022) conducted experiments with 21 participants to compare the effectiveness of Copilot paired with human programmers in terms of productivity and code quality. The results indicate that while Copilot can increase productivity by adding more lines of code, the generated code quality is lower due to the need to remove more lines of code during testing. Barke et al. (2023) observed 20 participants who collaborated with Copilot to complete different programming tasks in four languages, and found that the interaction with the programming assistant was bimodal in different collaboration mode. Bird et al. (2023) conducted three studies aimed at understanding how developers utilize Copilot. Their findings suggest that developers spent a lot of time assessing the suggestions generated by Copilot instead of completing their coding tasks. Peng et al. (2023) presented the results of a controlled experiment using Copilot as an AI collaborative programmer. They found that the experimental group who had access to Copilot completed tasks 55.8% faster than the control group. Zhang et al. (2023) investigated the programming languages, IDEs, associated technologies, implemented functionalities, advantages, limitations, and challenges when using Copilot. Vaithilingam et al. (2022) conducted a user study involving 24 participants to assess the usability of Copilot and its integration into the programming workflow. They found that while Copilot may not directly enhance the efficiency of completing programming tasks, it serves as a valuable starting point for programmers, saving time spent on searching. Liang et al. (2024) conducted a survey among software developers and found that the primary motivation for developers to use AI programming assistants is to reduce keystrokes, complete programming tasks quickly, and recall syntax. However, the impact of using these tools to help generate potential solutions is not significant. Gustavo et al. (2023) analyzed the use of Copilot for programming and compared it with earlier forms of programmer assistance. They also explored potential challenges that could arise when applying LLMs to programming. Ziegler et al. (2024) sought to assess the impact of Copilot on user productivity through a case study, aiming to align user perceptions with empirical data. Their research highlights the aspects in which Copilot has enhanced users' coding productivity and how it achieves these improvements.

6.3. Conclusive Summary

Most of the prior studies utilized controlled experiments or surveys to evaluate the effectiveness of Copilot. Our research is grounded in the perspective of software developers, focusing on the real-world problems they encounter when using Copilot, exploring the underlying causes and viable solutions. By analyzing the study results, we aimed to provide insights for Copilot users, the Copilot team, and researchers. Besides, we collected data from three popular

software development platforms and forums, i.e., GitHub Issues, GitHub Discussions, and SO, to ensure the comprehensiveness of our dataset.

7. Conclusions

In this study, we focused on the problems users may encounter when using GitHub Copilot, as well as their underlying causes and potential solutions. After identifying the RQs, we collected data from GitHub Issues, GitHub Discussions, and Stack Overflow. After manual screening, we obtained 473 GitHub issues, 706 GitHub discussions, and 142 SO posts related to Copilot and got a total of 1353 problems, 391 causes, and 497 solutions based on our data extraction criteria. The results indicate that *Operation Issue* and *Compatibility Issue* are the most common problems faced by users. *Copilot Internal Error*, *Network Connection Error*, and *Editor/IDE Compatibility Issue* are identified as the most common causes of these problems. *Bug Fixed by Copilot*, *Modify Configuration/Setting* and *Use Suitable Version* are the predominant solution. For the Copilot users, our results suggest that they should carefully review its code suggestions and seek inspiration from them. IDEs or code editors that are officially supported by Copilot can lead to a better user experience. For the Copilot team, it is essential to enhance the compatibility and provide support for a broader range of IDEs and code editors, simplify the configuration process, diversify the code suggestions while improving their quality, enhance Copilot performance for niche or domain-specific programming languages, and address concerns related to intellectual property and copyright. Additionally, users are asking for more customization options to tailor Copilot's behavior and more control over the code content generated by Copilot. For researchers, we found that additional time is required for code suggestion verification when utilizing Copilot, thus making code explanation feature especially valuable. Programming tasks in different application domains and the purpose for which users employ Copilot should be taken into consideration when assessing user satisfaction with Copilot. Consistently monitoring the feedback in *Feature Request*, particularly after introducing new features, can guide future improvements and identify specific areas for enhancing Copilot.

For future work, we plan to continuously collect new data from the three data sources to analyze the latest usage problems that occur during the iterations of Copilot, along with their causes and solutions. With the introduction of GitHub Copilot Trust Center (GitHub, 2024c), we plan to investigate whether the concerns of users for *Copyright and Policy Issue* will be addressed. Besides, we will employ an industrial survey to gather more information, such as the programming experiences of Copilot users and the development contexts when coding with Copilot, to further investigate how these factors impact the categories of Copilot-related problems encountered by users. Additionally, we aim to understand the impact of Copilot on various stages of software development life cycle by conducting an interview with developers who use Copilot in industrial software development. Besides, considering the emergence of LLMs is likely to drive a significant proliferation of AI code assistant tools, a comparison between Copilot and other popular AI code assistant tools (e.g., Tabnine (Tabnine, 2024), Amazon Q Developer (AWS, 2024)) holds valuable insights.

Data availability

We have shared the link to our dataset in the reference (Zhou et al., 2024).

CRediT authorship contribution statement

Xiyu Zhou: Conceptualization, Investigation, Data curation, Formal analysis, Writing - Original draft preparation. **Peng Liang:** Conceptualization, Methodology, Investigation, Data curation, Supervision, Writing - review and editing. **Beiqi Zhang:** Investigation, Data curation, Formal Analysis, Writing - Original draft preparation. **Zengyang Li:** Conceptualization, Methodology, Writing - review and editing. **Aakash Ahmad:** Conceptualization, Methodology, Writing - review and editing. **Mojtaba Shahin:** Conceptualization, Methodology, Writing - review and editing. **Muhammad Waseem:** Methodology, Writing - review and editing.

References

Akbar, M.A., Khan, A.A., Liang, P., 2023. Ethical aspects of chatgpt in software engineering research. IEEE Transactions on Artificial Intelligence .

Al Madi, N., 2023. How readable is model-generated code? examining readability and visual inspection of github copilot, in: Proceedings of the 37th International Conference on Automated Software Engineering (ASE), ACM. pp. 1–5.

Amin, M.M., Cambria, E., Schuller, B.W., 2023. Will affective computing emerge from foundation models and general artificial intelligence? a first evaluation of chatgpt. IEEE Intelligent Systems 38, 15–23.

Asare, O., Nagappan, M., Asokan, N., 2023. Is github's copilot as bad as humans at introducing vulnerabilities in code? Empirical Software Engineering 28, 129.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al., 2021. Program synthesis with large language models. arXiv preprint abs/2108.07732 .

AWS, 2024. Amazon Q Developer: The most capable generative AI–powered assistant for software development. https://aws.amazon.com/q/developer/?nc1=h_ls.

Barke, S., James, M.B., Polikarpova, N., 2023. Grounded copilot: How programmers interact with code-generating models. Proceedings of the ACM on Programming Languages 7, 1–27.

Bird, C., Ford, D., Zimmermann, T., Forsgren, N., Kalliamvakou, E., Lowdermilk, T., Gazit, I., 2023. Taking flight with copilot: Early insights

and opportunities of ai-powered pair-programming tools. ACM Queue 20, 35–57.

Campbell, J.L., Quincy, C., Osserman, J., Pedersen, O.K., 2013. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. Sociological Methods & Research 42, 294–320.

Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H.P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F.P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W.H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A.N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., Zaremba, W., 2021. Evaluating large language models trained on code.

Chen, X., Ye, J., Zu, C., Xu, N., Zheng, R., Peng, M., Zhou, J., Gui, T., Zhang, Q., Huang, X., 2023. How robust is gpt-3.5 to predecessors? a comprehensive study on language understanding tasks. arXiv preprint arXiv:2303.00293 .

Cohen, J., 1960. A coefficient of agreement for nominal scales. Educational and Psychological Measurement 20, 37–46.

de Dieu, M.J., Liang, P., Shahin, M., Khan, A.A., 2023. Characterizing architecture related posts and their usefulness in stack overflow. Journal of Systems and Software .

Eclipse, 2019. Eclipse Code Recommenders. https://projects.eclipse.org/projects/technology.recommenders.

El aoun, M.R., Li, H., Khomh, F., Openja, M., 2021. Understanding quantum software engineering challenges an empirical study on stack exchange forums and github issues, in: Proceedings of 37th IEEE International Conference on Software Maintenance and Evolution (ICSME).

Fu, Y., Liang, P., Tahir, A., Li, Z., Shahin, M., Yu, J., 2024. Security weaknesses of copilot generated code in github. arXiv preprint arXiv:2310.02059 .

Gartner, 2023. Gartner Identifies the Top 10 Strategic Technology Trends for 2024. https://www.gartner.com/en/newsroom/press-releases/2023-10-16-gartner-identifies-the-top-10-strategic-technology-trends-for-2024.

GitHub, 2024a. About GitHub Copilot Enterprise. https://docs.github.com/en/copilot/github-copilot-enterprise/overview/about-github-copilot-enterprise.

GitHub, 2024b. Configuring network settings for GitHub Copilot. https://docs.github.com/en/copilot/configuring-github-copilot/configuring-network-settings-for-github-copilot.

GitHub, 2024c. GitHub Copilot Trust Center. https://resources.github.com/copilot-trust-center//.

GitHub, 2024d. GitHub Copilot · Your AI Pair Programmer. https://github.com/features/copilot.

Gustavo, S., Hammond, P., Teo, N., Ramesh, K., Brendan, D.G., Siddharth, G., 2023. Lost at c: A user study on the security implications of large language model code assistants, in: Proceedings of the 32nd USENIX Security Symposium (USENIX Security), USENIX. pp. 2205–2222.

Huang, H., Wu, S., Liang, X., Wang, B., Shi, Y., Wu, P., Yang, M., Zhao, T., 2023. Towards making the most of llm for translation quality estimation, in: Proceedings of the 12th Natural Language Processing and Chinese Computing (NLPCC), Springer.

Imai, S., 2022. Is github copilot a substitute for human pair-programming? an empirical study, in: Proceedings of the 44th International Conference on Software Engineering (ICSE): Companion, IEEE. pp. 319–321.

Jaworski, M., Piotrkowski, D., 2023. Study of software developers' experience using the github copilot tool in the software development process. arXiv preprint abs/2301.04991 .

Jiang, J., Wang, F., Shen, J., Kim, S., Kim, S., 2024. A survey on large language models for code generation. arXiv preprint arXiv:2406.00515 .

Liang, J.T., Yang, C., Myers, B.A., 2024. A large-scale survey on the usability of ai programming assistants: Successes and challenges, in:

Proceedings of the 45th International Conference on Software Engineering (ICSE), ACM. pp. 1–13.

Luan, S., Yang, D., Barnaby, C., Sen, K., Chandra, S., 2019. Aroma: Code recommendation via structural code search. Proceedings of the ACM on Programming Languages 3, 1–28.

Mastropaolo, A., Pascarella, L., Guglielmi, E., Ciniselli, M., Scalabrino, S., Oliveto, R., Bavota, G., 2023. On the robustness of code generation techniques: An empirical study on github copilot, in: Proceedings of the 45th International Conference on Software Engineering (ICSE), IEEE. p. 2149–2160.

MITRE, 2023. CWE - Common Weakness Enumeration. https://cwe.mitre.org/index.html.

Moradi Dakhel, A., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M.C., Jiang, Z.M., 2023. Github copilot ai pair programmer: Asset or liability? Journal of Systems and Software 203, 111734.

Nguyen, N., Nadi, S., 2022. An empirical evaluation of github copilot's code suggestions, in: Proceedings of the 19th IEEE/ACM International Conference on Mining Software Repositories (MSR), IEEE. pp. 1–5.

Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., Karri, R., 2022. Asleep at the keyboard? assessing the security of github copilot's code contributions, in: Proceedings of the 43rd IEEE Symposium on Security and Privacy (S&P), IEEE. pp. 754–768.

Peng, S., Kalliamvakou, E., Cihon, P., Demirer, M., 2023. The impact of ai on developer productivity: Evidence from github copilot. arXiv preprint abs/2302.06590 .

Pope, T., 2024. Copilot.vim. https://github.com/github/copilot.vim.

Robillard, M., Walker, R., Zimmermann, T., 2010. Recommendation systems for software engineering. IEEE Software 27, 80–86.

Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14, 131–164.

Siddiq, M.L., Majumder, S.H., Mim, M.R., Jajodia, S., Santos, J.C., 2022. An empirical study of code smells in transformer-based code generation techniques, in: Proceedings of the 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM), IEEE. pp. 71–82.

Sobania, D., Briesch, M., Rothlauf, F., 2022. Choose your programming copilot: A comparison of the program synthesis performance of github copilot and genetic programming, in: Proceedings of the 24th Genetic and Evolutionary Computation Conference (GECCO), ACM. pp. 1019–1027.

Stol, K.J., Ralph, P., Fitzgerald, B., 2016. Grounded theory in software engineering research: a critical review and guidelines, in: Proceedings of the 38th International conference on software engineering (ICSE), ACM. pp. 120–131.

Tabnine, 2024. Tabnine - The AI code assistant that you control. https://www.tabnine.com/.

Vaithilingam, P., Zhang, T., Glassman, E.L., 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models, in: Proceedings of the 42nd CHI Conference on Human Factors in Computing Systems (CHI) Extended Abstracts, ACM. pp. 1–7.

Wang, C., Hu, J., Gao, C., Jin, Y., Xie, T., Huang, H., Lei, Z., Deng, Y., 2023a. How practitioners expect code completion?, in: Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FES), ACM. pp. 1294–1306.

Wang, R., Cheng, R., Ford, D., Zimmermann, T., 2023b. Investigating and designing for trust in ai-powered code generation tools. arXiv preprint abs/2305.11248 .

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E.H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., Fedus, W., 2022. Emergent abilities of large language models. arXiv preprint arXiv:2206.07682 .

Weisz, J.D., Muller, M., Houde, S., Richards, J., Ross, S.I., Martinez, F., Agarwal, M., Talamadupula, K., 2021. Perfection not required? human-ai partnerships in code translation, in: Proceedings of the 26th International Conference on Intelligent User Interfaces (IUI), ACM. p.

402–412.

Wilkinson, L., 2024. GitHub Copilot drives revenue growth amid subscriber base expansion. https://www.ciodive.com/news/github-copilot-subscriber-count-revenue-growth/706201/.

Yetistiren, B., Ozsoy, I., Tuzun, E., 2022. Assessing the quality of github copilot's code generation, in: Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE), ACM. pp. 62–71.

Zhang, B., Liang, P., Feng, Q., Fu, Y., Li, Z., 2024. Copilot refinement: Addressing code smells in copilot-generated python code. arXiv preprint arXiv:2401.14176 .

Zhang, B., Liang, P., Zhou, X., Ahmad, A., Waseem, M., 2023. Demystifying practices, challenges and expected features of using github copilot. International Journal of Software Engineering and Knowledge Engineering 33, 1653–1672.

Zhao, S., 2023. GitHub Copilot Chat. https://github.blog/2023-12-29-github-copilot-chat-now-generally-available-for-organizations-and-individuals/.

Zhou, X., Liang, P., Zhang, B., Li, Z., Ahmad, A., Shahin, M., Waseem, M., 2024. Dataset of the Paper "Exploring the Problems, their Causes, and Solutions of AI Pair Programming: A Study with Practitioners of GitHub Copilot". https://doi.org/10.5281/zenodo.11080113.

Ziegler, A., Kalliamvakou, E., Li, X.A., Rice, A., Rifkin, D., Simister, S., Sittampalam, G., Aftandilian, E., 2024. Measuring github copilot's impact on productivity. Communications of the ACM 67, 54–63.