# Studying the effect of GPT-3 Technology (Copilot) on User Trust

*Nephele Aesopou*


4th Year Project Report
Computer Science and Mathematics
School of Informatics
University of Edinburgh

2024

# Abstract

Automatic code generation and facilitation of programming tasks has long been a key focus for developers and researchers, seeking to streamline such processes and increase productivity. Despite the growing adoption of AI-driven code generation tools, the evaluation of such tools often overlooks the critical aspect of human-computer interaction in educational settings. Specifically, users' trust in AI coding assistants is crucial for their effective adoption and integration into software development processes and education.

This thesis addresses this gap by investigating how students adopt and use generative AI tools for coding tasks. We conducted an extensive user study involving university computer science students focusing on the trust they develop towards GitHub Copilot, an AI code generation tool. The study measured trust quantitatively, through metrics like the acceptance of Copilot's suggestions, and qualitatively, via questionnaire responses. Additionally, we examined the influence of students' initial expectations, perceived productivity, and task complexity on their trust towards Copilot, aiming to understand how different factors affect reliance on the tool.

This study revealed two distinct interaction patterns among students, *Collaborators* and *Operators*, highlighting the contrasting approaches to leveraging AI assistance. Furthermore, it provided insights into factors impacting user trust, such as met expectations and perceived productivity, while underscoring the importance of responsible AI usage in educational settings to avoid over-reliance.

# Research Ethics Approval

This project obtained approval from the Informatics Research Ethics committee.
Ethics application number: 676940
Date when approval was obtained: 2023-12-08
The participants' information sheet and a consent form are included in Appendix B.

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Nephele Aesopou*)

# Acknowledgements

I would like to express my sincerest gratitude to my supervisor, Dr Kobi Gal, for his invaluable guidance, constructive feedback, and continuous encouragement throughout the project. Without him I would not have had the chance to explore the interesting world of generative AI programming assistants, and learn the steps towards the creation of an insightful user study.

I am indebted to Professor Fiona McNeill for generously allowing me to collaborate with her course “Introduction to Object Oriented Programming”, use the class materials, and for sharing my passion for effective and responsible use of Artificial Intelligence in the educational setting.

I would also like to give many thanks to all participants of my user study, who dedicated their time (and patience) to working with Copilot. Their contributions were invaluable to this project.

Finally, I am grateful to my friends and family for their unwavering support, endless motivation, and understanding throughout this challenging journey.

# Table of Contents

# Chapter 1

# Introduction

## 1.1 Background and Motivation

The programming landscape is undergoing a significant transformation with the emergence of large pre-trained models designed to automate code generation [12, 11]. These systems aim to increase programmer productivity, automate routine tasks, and facilitate the process of acclimating users to other languages or novel codebases [24, 41, 11]. As such tools become widely accessible and integral to software development, it is crucial to examine their capabilities and impacts.

A prominent example of such technology is GitHub Copilot, an Artificial Intelligence (AI) tool that significantly streamlines code development [2]. Copilot suggests lines of code or entire functions by analysing context from code snippets and comments, showcasing its proficiency in understanding both programming and natural languages. This capability stems from its training on a vast corpus of publicly available source code and documentation, highlighting its potential to transform programming practices and educational methodologies alike [11].

While existing research primarily focuses on the accuracy, productivity, and security of these AI systems, it overlooks the importance of trust in the human-AI collaboration and the way students interact with and perceive AI coding assistants [11, 17, 24, 25, 41]. This thesis extends the existing research by presenting the first study centering on university students and their development of trust in Copilot through an extended period of coding tasks, in contrast with previous one-off evaluations. The longitudinal approach is motivated by the need to explore the dynamics of trust between students and AI tools over time, investigating the conditions under which users trust AI-suggested code. The development of trust in AI models is pivotal, influencing how much responsibility and tasks humans are willing to delegate to these tools [35]. This aspect is especially critical considering the future landscape of jobs like software and data engineering, which will increasingly rely on strong AI capabilities. Furthermore, the impact on education is profound, raising questions about how to effectively integrate AI coding assistants while ensuring that students develop robust problem-solving skills.

## 1.2 Objectives and Research Questions

The overall goal of this thesis was to study the effect of Copilot on student trust and generate useful insights from the interactions between the students and AI tool. This project is centred around the following three research questions:

- **RQ1:** How do students specify queries to Copilot, and how do these queries evolve over time? Additionally, how do users' interactions with Copilot adapt in response to its behavior?
- **RQ2:** How to define and measure user trust, as it may be reflected in students' interactions with Copilot?
- **RQ3:** How do factors such as task complexity, students' perceived productivity, and their initial expectations of Copilot influence their trust in the tool?

## 1.3 Methodology Overview

To address the research questions, we designed and executed a user study with eight university students in collaboration with an introductory Java university course (Inf1B). This study comprised of four self-contained Java programming tasks from the Inf1B lab and tutorial exercises, aimed at introducing students to basic Java concepts. Participants were encouraged to code with the assistance of GitHub Copilot.

Moreover, participants completed three surveys: the first gathered their background information and initial expectations of Copilot, while the remaining two focused on their reflections after the interaction. The study spanned two weeks, with participants completing one task during the first week and three tasks during the second. This schedule allowed for an analysis on how interactions changed and how trust in the tool evolved as participants became more familiar with both the programming language and Copilot.

Participants were instructed to screen-record their coding sessions. These recordings were later reviewed manually by the researcher. The analysis focused on several criteria, including measurement of acceptance of Copilot's suggestions, any subsequent modifications or deletions of these suggestions, and the categorization and frequency of user prompts and repair strategies. Additionally, a comprehensive analysis of survey responses from all participants was performed. Further details on the user study methodology are elaborated in Chapter 3.

## 1.4 Key Findings and Contributions

Our main finding is the identification of two distinct interaction patterns among students when incorporating AI code-generation tools for solving programming tasks: the *Collaborators* and the *Operators*. These coding styles relate to the type of prompting strategy followed by users and the level of coding they perform. Collaborators have a clear problem-solving approach and guide Copilot's suggestions, while Operators focus on writing natural language instructions to the AI through prompts and on editing

the given solution only when it provides errors. The study also revealed that unmet initial expectations, such as Copilot's code debugging abilities, significantly decreased trust levels. Notably, we observed that perceived productivity remained strong, despite instances of long debugging sessions.

This thesis presents the first empirical longitudinal user study analysing trust development between undergraduate students and AI coding assistants like GitHub Copilot. A key contribution is the identification of two contrasting interaction styles, which can inform the design of AI tools and their adoption in educational settings, as well as the need to avoid over-reliance. The quantitative and qualitative analysis of programming interactions sheds light on factors influencing trust in AI tools. Furthermore, the prompt categorisation observations and analysis offer insights into the prompting strategies employed by students. Importantly, our findings highlight the necessity of guiding students to leverage AI tools responsibly without excessive dependence, which could hinder their problem-solving abilities.

## 1.5 Structure of thesis

This thesis comprises of six chapters:

**Chapter 1:** presents the background and motivation for the study and defines the project objectives and research questions. It also gives an overview of the study's methodology and its key findings and contributions.

**Chapter 2:** delves into a quick summary of related literature, looking into Large Language Models, GPT-3 technology, and relevant user studies. The examined papers are divided into four categories depending on the topic of their research: user trust, security, productivity, and usability and correctness.

**Chapter 3:** describes in detail the user study design and methodology. Specifically, it defines user trust between AI and programmer, justifies the design of the programming scenarios and the selection of participants, and thoroughly explains the evaluation criteria.

**Chapter 4:** presents the empirical results of the user study, dividing them into programming task results and survey responses. The analysis is done both quantitatively and qualitatively.

**Chapter 5:** discusses the results of the analysis, highlighting two distinct participant interaction patterns: *Collaborators* and *Operators*. Moreover, it elaborates on the results of acceptance metrics, perceived productivity, over-reliance, and education.

**Chapter 6:** gives a summary of the thesis' findings and the answers to the three research questions. It concludes with the study's limitations and suggestions for future work.

# Chapter 2

# Related Literature

This thesis relates to work in several areas, including Large Language Models (LLMs), Generative Pre-trained Transformers (GPTs), and specifically focuses on studies relevant to AI code-generation assistants and research on user trust. Each area is expounded upon, discussing its connections to the three research questions.

## 2.1 Large Language Models

Generative AI has undergone remarkable advancements in recent years, leading to a surge in its popularity and revolutionizing problem-solving and content creation [28, 5]. Generative AI systems can generate a diverse range of outputs, including text, images, speech, and code, that closely resemble human-generated content [18, 6]. LLMs are characterized by deep neural network architectures trained on vast textual data comprising of billions of parameters [21]. By sequentially analysing data and tracking relationships in their word corpora, LLMs can generate coherent and contextually relevant text from small natural language prompts, demonstrating remarkable utility in Natural Language Processing [28, 18, 5].

## 2.2 GPT-3, Codex & Copilot

OpenAI has significantly contributed to the development of LLMs by introducing "Generative Pre-trained Transformers" in 2018 [27]. Their subsequent GPT-3 [10] marked a ground-breaking achievement in the domain of unsupervised neural networks in 2020 and formed the foundation for the Copilot GPT model. The following subsections analyse the performance of GPT-3 and two of its extensions: Codex and Copilot.

### 2.2.1 GPT-3 and Few-shot Learning

Studies have evaluated GPT models' performance, investigating how outputs can be influenced by the choice of prompts and training data [38, 37, 40, 19]. Notably, Zhao et al. [38] revealed that GPT-3's accuracy was heavily susceptible to the prompt format, the set of training examples, and their ordering. To overcome the biases, the authors

proposed “Contextual Calibration”, a procedure to adjust the model’s output probabilities and correct the contextual error. This result highlighted the importance of prompt engineering for AI code generation tools like Copilot [37, 40, 19]. Nevertheless, in the case of Copilot, an explicit prompt is not always necessary, as the tool can leverage information from the coding context and provide implicit suggestions [39].

### 2.2.2 Codex

To explore GPTs’ language understanding capabilities, Chen et al. [11] fine-tuned the pre-trained GPT-3 language model on millions of public software repositories hosted on GitHub, creating Codex, a large-scale transformer model focused on program synthesis. Codex was trained on 159GB of filtered public python code and can generate python functions to solve problems given by natural language prompts [11]. Its performance was evaluated on HumanEval, a dataset of 164 hand-written problems which assessed language comprehension, reasoning, simple mathematics, and algorithms.

The authors discussed Codex’s limitations, which include sample inefficiency, over-reliance, alignment failure, biased code due to the collection of human-generated programs, cybercrime risks, and a huge environmental impact [11].

OpenAI has since granted selective public access to its models through tools like ChatGPT and Copilot [2, 1]. ChatGPT for instance, serves as a user-friendly interface that allows AI-driven conversations. In parallel, OpenAI’s collaboration with Microsoft GitHub led to the development and launch of Copilot in October 2021 [2].

### 2.2.3 AI pair-programmer: Copilot

Copilot, a model powered by Codex, is introduced as the “AI pair-programmer”v [2]; an AI tool designed to assist the programmer in real-time. Users can simply articulate their coding intent by providing a natural language comment, docstring description, or by specifying a function name and its associated arguments (see Figure 2.1). Copilot, in response, instantaneously suggests relevant code.

The programmer can then accept the suggestion using the `Tab` key, request an alternative with `Alt + ]`, or discard Copilot for this specific task by pressing `Esc`. Furthermore, Copilot features a multi-lined suggestion pane that permits users to view and compare diverse implementations of the same function and is able code in multiple programming languages.

Finnie-Ansley et al. [17] evaluated Copilot’s accuracy by comparing its performance on introductory computing exercises to students completing the same exercises in an examination setting. Copilot was presented with 23 student exam questions and set to solve them using the same examination software and grading scheme. Additionally, it was presented the Rainfall problem [30] with wording variations, and its passing rate was measured by testing each problem 50 times. Codex scored around 75% on the exam exercises, placing it among the top quarter of the students’ scores. The different Rainfall solutions generated by Copilot were examined by counting lines of code and the algorithmic variation [17].

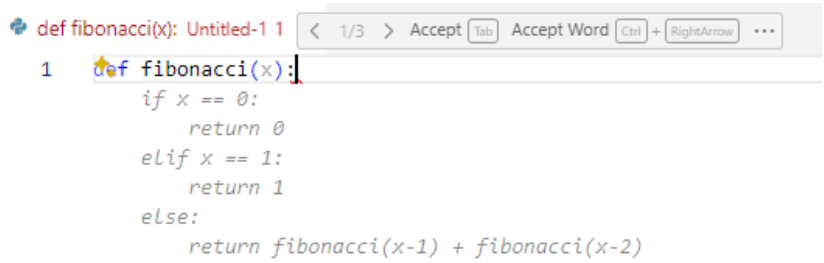


Figure 2.1: Interaction with GitHub Copilot, showing the user's code and Copilot's suggestion in gray.

While our user study also selected a set of coding tasks for novice programmers, we expand the research by studying the interaction between the students and the AI tool, and by discussing how this might affect education and learning practices.

## 2.3 User studies on Copilot

Despite its impressive accuracy, there exists limited research on Copilot, whether in term of its capabilities other than accuracy, or from the user's perspective. Specifically, one must consider user trust [35], security [23, 25, 29], productivity [41, 24], and usability [31, 8, 11].

### 2.3.1 User Trust

User trust is crucial for an automated code generation tool like Copilot [13]. This thesis expands upon the limited existing research on user trust towards AI code-generation tools, while drawing comparisons with important observations concerning the establishment and maintenance of trust.

To measure user trust, different approaches have been explored, such as the work by Perry et al. [25], where trust was correlated with survey responses, free-response feedback, and the uptake of AI suggestions. They concluded an inverse relationship between security concerns and trust in the AI assistant. Building upon these measures, our study adapts them to an undergraduate programming environment. Moreover, in order to analyse the users' approach to learning how to trust Copilot, Perry et al. [25] defined twelve prompt categories for prompts provided to the AI assistant. Users appeared to "learn" the best strategy to address the AI and how to re-phrase their prompts for the desired output. We adapt Perry et al. [25]'s analysis approach to the observations from our study's programming task screen recordings.

Contrary to other studies, Wang et al. [35] avoided the controlled task completion and interviewed professional developers interacting with AI assistants in their real-time jobs [35]. The two-stage analysis involved the interview stage and a design probe investigation, whose goal was to interpret the design choices that contribute to trust.

According to Wang et al. [35]'s results, what contributes most to the user's trust is the expectation of the AI's capabilities, such as productivity, higher code quality, knowledge of AI's limits, and awareness of its potential risks. Furthermore, authors identified the challenges in managing user expectations effectively (tiresome learning process, few use-cases, biased towards other AI tools), and the difficulties in regulating Copilot and evaluating its suggestion. Contrary to our analysis, Wang et al. [35]'s study predominantly focused on the effect of the user interface design on the coding experience. Finally, our user study also faces the limitations of gender and self-selection bias, as participants who volunteered were interested or experienced with such AI tools [35].

### 2.3.2 Security

A prominent 2022 user study implemented by researchers at Stanford University, measured the security risks that arose when writing code with an AI assistant [25]. Their goal was to examine the relationship between AI usage, the rate of security mistakes, and users' different prompting strategies. The study consisted of a set of self-contained, short tasks in multiple programming languages, that covered a wide range of potential security mistake types. Using A/B Testing, participants were divided into an Experiment group with access to the AI assistant (2:1 ratio) and a Control group without access. The overall result showed that users with access to the AI wrote incorrect and insecure code more often than the Control group, with strong statistical significant results for half of the tasks and marginal significance for the rest.

Aligning with the focus of this thesis, Perry et al. [25]'s user study primarily enlisted participants from the university student demographic (66% of participants). However, the authors remarked that this demographic may not provide a very descriptive representation of the population that engages daily with such AI tools and security issues. Unlike our work where interactions were done in an integrated development environment, the study by Perry et al. [25] implemented a programming interface that required actively prompting the AI model. Moreover, their analysis focused on the correlation between trust and security risks, compared to our broader investigation of trust in a university course setting.

### 2.3.3 Productivity

Using a similar approach, Peng et al. [24] assessed whether Copilot increased human productivity [24]. Specifically, their controlled experiment recruited 95 professional software engineers and asked them to write a toy HTTP server engine in JavaScript as quickly as possible, using A/B testing. The results of the experiment showed an 55.8% productivity increase, which was measured as the average completion time of participants using Copilot minus the average completion time of the participants without assistance [24].

Interestingly, the study was conducted before the release of the Copilot tool, so participants were given an introductory tutorial before attempting the problem. All participants had no experience with the AI tool and their expectations of Copilot were unbiased.

Similar to our user study, results were affected by participants' inexperience with Copilot and by the time and effort they required to discover its capabilities and learn how to use it. We resolve the issues of a one-off interaction by conducting a longer duration study, spanning two weeks.

Since productivity was measured by task success and task completion time, factors such as task experience, familiarity with HTTP, and participant dedication, might have influenced the results. Investigating the demographics, the researchers suggested that "less experienced developers, developers with heavy coding load, and older developers benefit more from Copilot" [24].

### 2.3.4 Usability and Correctness

Questions like "How do users recognize errors in code generated by Copilot?", "What coping mechanisms do users employ when they find errors in code generated by Copilot?" and "What are the obstacles and limitations that can prevent adoption of Copilot?" were investigated by Vaithilingam et al. [31] in 2022.

The study used Copilot to discover programmers' expectations, coping strategies, and needs regarding an AI code-generator assistant [31]. It recorded the performance of 24 participants completing three Python coding tasks of increasing difficulty, once using Intellisense (default completion tool in VS Code [4]) and once using Copilot. The tasks were considered failed if not completed withing the 20 minute time limit. The researchers collected data on task completion time, failure rates, and survey response metrics during and after the programming tasks.

A non-statistically significant but important result was that participants using Copilot failed more tasks than when not, but successful tasks were completed faster with Copilot. The authors highlight their observation that users underestimated the effort required to fix errors in the suggested code, leading to a significant debugging time and task failure.

Unlike our user study involving undergraduate students (most of them new to Copilot), Vaithilingam et al. [31] recruited participants with advanced programming experience, with less than half being undergraduates.Their study generated very interesting observations regrading users' coping strategies and obstacles, which we compare in this thesis with our student population results.

Barke et al. [8] analysed Copilot's usability and the interactions between programmers and AI assistant based on Grounded Theory; a qualitative research technique that iterates between data collection and theory hypotheses [8]. By asking the participants to complete coding tasks across different programming languages, the authors concluded two types of user interaction: "Acceleration mode", where coders know their next steps and interactions are fast and short, and "Experiment mode", where the programmers are unsure how to proceed and prompt the AI to get assistance and ideas [8].

The authors suggest that the programming assistant should identify the interaction mode and adjust its suggestion type, frequency, and confidence level accordingly. We explore if these interaction patterns apply to undergraduate students, since Barke et al. [8] mostly recruited participants from academia. We extend the investigation of interaction

patterns, by setting a longer study duration to allow users to familiarise themselves with Copilot without the time pressure.

Focusing on usability, the user study by Prather et al. [26] explored the interaction between university students and Copilot from a teaching and learning perspective rather than user trust. Students were presented with a programming homework assignment and allowed to use Copilot. Different to our user study, it was a one-off interaction in an examination environment, potentially affecting students' performance due to stress. The paper never measures student trust towards Copilot. Prather et al. [26] identified two distinct interaction types: "Shepherding", where unsure students got distracted by Copilot's suggestions and tried to guide it towards the correct solution, and "Drifting", where students entered a debugging "rabbithole" and wasted a lot of time in trying to understand suggestions. Both interaction patterns, common among novice programmers and new Copilot users, were observable in our user study.

# Chapter 3

# User Study Design

The design and methodology of this project aligns with previously published research on GitHub Copilot, adopting similar approaches for task selection, data collection techniques, and analysis methods [25, 31, 8, 26]. Specifically, we designed a user study that combines programming tasks with three surveys.

Programming tasks were performed by the study's participants with the help of GitHub Copilot and the screen-recording of each interaction was analysed manually by the researcher using the evaluation criteria described in Subsection 3.5.2. Additionally, a thorough analysis of survey responses was conducted to pinpoint recurring themes and patterns in participants' perceptions and comments. The focus of the analysis was on the level of trust in Copilot, the factors that influence user trust, and the users' behavior during the interaction.

Due to the various definitions that the term *trust* can adopt and the multifarious possible methods that such a user study could follow, in the following chapter we state and justify the decisions made regarding the definition of trust, the programming tasks, the participant pool, and the methodology.

## 3.1 Definition of User Trust

The most important task was to define *trust* between humans and AI. One possible approach to defining trust is through the seven key requirements published by the European Commission in 2019, which outline the criteria for an AI system to be considered trustworthy [13]. These requirements include human agency and oversight, technical robustness and safety, privacy and data governance, transparency, diversity and non-discrimination and fairness, societal and environmental well-being, and accountability.

Applying these criteria to GitHub Copilot, previous studies, such as the one by Pearce et al. [23], have focused on the security and safety aspect of trust. Others, evaluate Copilot's transparency in explaining the rationale behind its actions and decisions, as users require accountability and feedback [34]. Additionally, user experience plays a vital role in developing trust, as a smoother and more positive experience can enhance the tool's trustworthiness, an aspect investigated by Wang et al. [35].

It is evident that trust cannot be captured through a singular measure and is highly situation-dependent. It is also characterized by its qualitative and dynamic nature, being influenced by the outcomes of interactions with the AI. This study aims to analyse the change in users' attitude and trust towards the AI, acknowledging that trust entails maintaining "positive expectations while accepting the perceived risks of undesirable outcomes" [35].

### 3.1.1 The importance of Trust between users and AI

Why is trust in AI tools critical today? Firstly, it drives their adoption and frequent use. The success of an AI tool depends on its users' trust Wang et al. [35]. This trust influences users' willingness to utilize AI models in unfamiliar tasks. For instance, in education, students are more likely to incorporate AI code-generating tools into their daily programming tasks if they trust their capabilities. Secondly, trust is closely tied to responsible AI usage. For example, when students trust the AI, they implicitly assume responsible usage, contributing to the overall responsible deployment of AI.

In addition, it is crucial to be able to measure this trust due to the numerous risks associated with blindly relying on an AI tool. Over-reliance on AI models can result in numerous issues including security breaches, lack of diversity, violation of creator rights, and impediment of educational progress [25, 11, 26]. We discuss the topic of over-reliance in depth in Section 5.5. Conversely, completely distrusting the AI in today's context is simplistic, as AI holds potential benefits for the society and economy. This project aims to show the importance of trust between the user and AI tool, and investigate how it is established, maintained, or lost.

In the first survey of the study (refer to Appendix Section A.2), two questions addressed the topic of trust by asking the participants to select the option with the greatest and least impact on building trust between users and AI. Interestingly, all participants believed that the most important factor is the accuracy of the AI tool. Regarding the factor with the least impact on trust, there was a varying opinion: most participants chose Ethical behavior of the AI tool (50%), while others voted for User experience (25%), Consistency of suggestions (12.5%) and Data Privacy & Security (12.5%), as discussed in Section 4.2.

### 3.1.2 User Trust in this study

In this thesis, we chose to determine trust in terms of measure of acceptance of Copilot's suggestions, and from the analysis of the programming interactions and free-response questions. Specifically, we analysed the number of accepted Copilot suggestions, as well as if they led to subsequent edits or deletions. Moreover, since trust is a theoretical concept, we measured it through survey responses focusing on students' perceptions and expectations.

## 3.2 Programming scenarios design

The second critical decision in planning the user study was designing the programming scenarios. Decisions concerning the type of coding tasks, their difficulty, the study's duration, and the implementation of A/B testing, required careful consideration both individually and collectively due to their interdependence.

### 3.2.1 Inf1B course

We chose to focus the user study on programmers that have different levels of experience with Copilot and the chosen coding language. This decision was made to try and separate the user's experience with solving a certain task and the trust he has in Copilot. In other words, if a user is very familiar with the coding task he is given, it will be easier for him to understand and accept/edit Copilot's suggestion, without that indicating his high trust to the tool. On the other hand, novice and intermediate programmers that are confronted with tasks they have not frequently seen before, are probably more likely to trust Copilot and will show stronger adoption changes as they interact for longer. Therefore, the study was done in collaboration with the instructor of the "Introduction to Object Oriented Programming" (Inf1B) course, a foundational Java course at the university. The recruitment of participants was targeted at first-year University of Edinburgh computer science students. As a result, our study primarily centered on tasks aligned with the introductory nature of the course, catering to the skill level of first-year students.

### 3.2.2 Task programming language

The Java programming language was selected for two main reasons: firstly, the Inf1B course presented the ideal environment for the programming tasks, and secondly, it is currently one of the top three most commonly used languages in GitHub, hence Copilot is very familiar with it. Our programming tasks were done in one programming language, similar to other relevant studies [26, 24, 31]. We selected this approach because we wanted to focus our investigation around trust that should not be affected by the programming language. We believe that our results would be the same irrespective of the coding language.

### 3.2.3 Task duration

The user study spanned two weeks, coinciding with Weeks 2 and 3 of the University of Edinburgh's second semester, a period during which students were still acclimating to Java and motivated to participate in our study. Task 1 was assigned for completion in Week 1, and Tasks 2 to 4 in Week 2. We believe that assessing user trust in an AI tool based on a single interaction is insufficient, as trust may change across interactions and tasks. By extending the study over two weeks, users had ample opportunities to explore the AI's capabilities and evaluate its trustworthiness. Furthermore, this extended time-frame allowed us to examine how users adjusted their prompt format and refined their expectations, while becoming more familiar with the AI tool and as the tasks grow in complexity [25].

### 3.2.4 Programming Tasks

Users were asked to complete four self-contained programming tasks, one during the first week of the study and three during the second week. The tasks were exercises taken from the tutorial and lab sessions of the Inf1B course, as they had the appropriate difficulty level and did not create additional work for the students. All four tasks could be completed in a short amount of time and were integral for mastering Java. Participants could use the internet and any documents (such as course materials), as they would have done normally when solving the class exercises.

We present below the four programming tasks assigned to the participants. Task specifications are copied and shortened from the Inf1B tutorial and lab sheets. These tasks primarily evaluated proficiency in basic Java functions, for-loops, and overall code planning skills.

1. **Task 1: Esrever (Reverse)**

   Given a string containing a sentence, how would you output the sentence with the order of words in reverse? For example, given "I like ice cream" the output should be "cream ice like I".

2. **Task 2: ArrayRotate**

   In this exercise, the goal is to write program called ArrayRotate that takes a series of integers from the command-line, stores them in an array *nums*, then copies them into a new array *copy* so that the values are rotated left by one.

3. **Task 3: Mode**

   Write a program Mode to produce a table of the results, specifying the number of instances of each value, followed by the corresponding number of dots, as a simple visualisation. Finally, your program should print out the mode.

4. **Task 4: Sieve of Eratosthenes**

   Write a program Sieve which finds all the prime numbers up to 20 using the Sieve of Eratosthenes (checking that $p < n$ rather than $p < n^2$) and prints them to the terminal on a single line.

Spanning from Week 1 to Week 2 exercises, the tasks progressively increased in complexity. They served to reinforce comprehension of prior topics while facilitating the expansion of participants' knowledge.

### 3.2.5 Task time limit

We followed a different approach from other studies [26, 31, 25] and set no time limit on the programming tasks to replicate as faithfully as we could the learning environment of Inf1B class. As we highlighted to the students, their goal was to interact and explore the capabilities of Copilot without the pressure of a time limit.

### 3.2.6 A/B Testing

We opted against employing A/B testing for our user study to diverge from the methodology used in [25, 24], believing it to be less effective for assessing trust in Copilot within a university context. A/B testing would focus on making performance comparisons between the two groups and it would limit the number of participants interacting with Copilot. Consequently, we chose to have all participants engage with the same coding tasks using Copilot, aiming for a more focused analysis of trust dynamics.

## 3.3 Participant pool

In order to satisfy the decisions taken regarding the programming tasks and their difficulty, the user study was aimed at first year undergraduate computer science students. There were no requirements to participate other than a keen interest in AI code generation tools and in discovering Copilot's capabilities.

### 3.3.1 Greatest Challenge: Recruitment of participants

One of the biggest challenges of the study was recruiting participants. Due to the nature of this study and its length spanning two weeks, it was difficult to continuously motivate the participants to submit their Copilot interactions. Moreover, for many novice programmers, recording their screen while new to coding was a stressful environment, even thought it was highlighted that the study was independent from the Inf1B course. Finally, we were unable offer any benefits to the participants as compensation for their time, as done by other user studies [31, 24, 25].

### 3.3.2 Study participants

Initially, 15 students signed up for the user study. However, not all of them continued to participate and did not upload the recordings of their interactions with Copilot. In order to recruit more participants, friends and university contacts of the researcher were prompted to participate. This project was thus completed with 8 participants. The detailed summary of the participant demographics is given in Table A.1 in Appendix Section A.3. We believe that the number of participants enrolled in the user study was adequate to conduct the research and gain insightful answers to our research questions. Similar user studies had 10 - 24 recruited participants, due to the large amount of manual work required to review the interactions between the participants and AI [31, 8, 35, 26].

We asked the participants of their overall programming experience (5 intermediate and 3 advanced) as well as their Java programming experience (5 beginners and 3 intermediate) in the first survey (see Appendix Section A.2). We also set a question about their familiarity with Copilot and how often they use it. From the eight participants, seven had heard of Copilot before, while only one of them was using it daily. Hence, for most of the participants this was their first interaction with Copilot, similar to participants in Prather et al. [26]'s study.

## 3.4 Ethics

This project obtained approval from the Informatics Research Ethics committee. The only personal data collected was the participant student number to further communicate the installation and task instructions on Copilot (see Appendix Section A.1), and to create a protected folder to share the screen recordings. Afterwards, all data was pseudonymised. We informed students that everything would stay anonymous and that the correctness and efficiency of their code would not be associated to them or their performance in the Inf1B course.

## 3.5 Methodology

In this section we present the methodology followed by the user study. Firstly, we explain how this study was orchestrated and give the general timeline. Then, we define and describe in detail the evaluation criteria we used to analyse the programming tasks and the responses to the three surveys.

The analysis phase presented significant challenges, mainly due to the large volume of data: 218 minutes of screen recordings that needed detailed review. Understanding each participant's programming style and defining precise evaluation metrics for the coding tasks, added complexity to the analysis. This required meticulous and slow measurement of the metrics for each screen recording to ensure accuracy and reliability of our analysis.

### 3.5.1 Preparation and Execution of the User Study

Initial contact with participants occurred in the beginning of the second semester, when the study was introduced during a lecture to motivate student participation. To increase participation numbers, an email was sent to the Inf1B class cohort, detailing the research's significance, and promising to share study results with participants.

Participants were required to sign-up in the study by completing Survey 1 (see Appendix Section A.2), which included the Participant Information Sheet and Consent Form (see Appendix Section B) along with 14 background questions covering demographics, programming experience, and views on user trust and Copilot. Participants then received instructions via email for accessing GitHub Copilot through GitHub Education (free access for students) and for installing Copilot on their preferred Integrated Development Environment (see Appendix Section A.1). The instructions also provided a brief introduction to Copilot, outlined the study tasks, and explained how to share screen recordings with the researcher. Furthermore, they contained links to two concise surveys (Surveys 2 and 3, see Appendix Section A.2) to be completed each week, featuring questions like "Did you trust Copilot for your answer?", "Did your trust for Copilot change after this interaction?" and "Is there a feature you would add/remove from Copilot?".

| Metric | Variable | Description |
|---|---|---|
| Coding minutes | `time_min` | Active programming minutes, rounded. |
| Coding seconds | `time_secs` | Active programming seconds from coding start to stop. |
| Suggestion acceptances | `accepts` | Times a Copilot suggestion was accepted (`Tab`). |
| Suggestion deletions | `deletes` | Times a Copilot suggestion was deleted post-acceptance. |
| Suggestion edits | `edits` | Times a Copilot suggestion was edited post-acceptance (If user accepts only next word/line of the suggestion it is measured as an `edit`). |
| Suggestion rejects | `rejects_I` | Times a Copilot suggestion was read and rejected either by typing or by `Esc` key. |
| Queries to Copilot | `no_queries` | Number of queries made to Copilot through comments (e.g. `// Write a for-loop`). |
| Alternative suggestions | `other` | User requested an alternative suggestion from Copilot (`Alt + ]`) |
| Internet use | `internet` | User accessed the Internet. |
| Code correctness | `correct` | Final solution correctness. |

Table 3.1: Task evaluation metrics.[1]

### 3.5.2 Evaluation Criteria

In order to proceed with the analysis, it was necessary to define several assessment metrics, or criteria that we would use to evaluate each task. We split the evaluation criteria into four main groups: overall programming task metrics, prompt categorisation metrics, repair categorisation metrics, and survey evaluation criteria. We describe in detail the metrics in each of these groups and we present the results in Chapter 4.

#### 3.5.2.1 Overall programming task metrics

The overall task metrics included measurements of the correctness of the solution, acceptances of Copilot's suggestion and time completion of each task. We give a detailed list of these metrics in Table 3.1.

#### 3.5.2.2 Prompt Categorisation

Inspired by the prompt analysis conducted by Perry et al. [25], we used a similar approach to categorize prompts provided to Copilot during the interactions. We investigated how users specify queries to Copilot and if they change over time and how the user's choice of prompt type influences their trust in Copilot. We decided to distinguish nine different prompt types that we explain in Table 3.2.

[1] All binary metrics are marked as 1 for "Yes" and 0 for "No".

| Prompt Category | Variable | Description |
|---|---|---|
| Function Declaration | `prompt_fd` | Declares a function with parameters (e.g. `public main String reverse(String input)`). |
| Variable Declaration | `prompt_vd` | Declares a variable with its type and name (e.g. `int[] count`). |
| Method Declaration | `prompt_md` | Declares method for next action (e.g. `for`, `while`, `System.out.println()`. |
| Implicit Suggestion | `prompt_none` | User skips to next line and gets an implicit suggestion. |
| Instructions | `prompt_instruct` | User instructs Copilot via a comment (e.g.`//Use while loop to check p<n`). |
| Short Instructions | `prompt_short` | Instructions less than 50 characters. |
| Long Instructions | `prompt_long` | Instructions more than 50 characters. |
| Text Close | `prompt_text` | Directly uses task instructions as the prompt. |
| Library | `prompt_lib` | Import statement for libraries (e.g., `import java.util.*;`). |

Table 3.2: User Prompt Categories

#### 3.5.2.3 Repair Categorisation of programming tasks

We aimed to analyse how the participants' prompts mature as they interact with the AI pair-programmer. A query repair is "the gradual refinement of a prompt to optimise for the system output" [25]. By investigating the users' different repair strategies we intended to find the most common repair strategy across the overall study. We define the four repair types in Table 3.3.

| Repair type | Variable | Description |
|---|---|---|
| Expand Scope | `repair_expand` | Increase information or prompt size to refine suggestions. |
| Reword | `repair_reword` | Modify or reorder words without changing prompt length. |
| Continue | `repair_cont` | Continue typing to provide additional information and elicit new suggestions. |
| Retry | `repair_retry` | Retry the same prompt to receive a different suggestion. |

Table 3.3: User Repair strategies

#### 3.5.2.4 Questionnaires analysis

We analyzed the participant questionnaires through both quantitative and qualitative methods, linking their responses directly to the evaluation of their programming interactions with Copilot. Additionally, an aggregate analysis was conducted to achieve a comprehensive understanding of user trust in the GPT-3 tool.

# Chapter 4

# Empirical Results

This chapter presents the empirical results obtained from the programming tasks and survey responses of the user study. We divide our analysis into two sections: programming task results and survey response analysis.

The programming task results section reports quantitative metrics, including the acceptance metrics of Copilot's suggestions and the distribution of prompts across different categories. Moreover, we discuss qualitative observations from these interactions. The survey response analysis section examines participants' responses to both structured and open-ended questions, providing insights into their initial expectations, perception of trust towards generative AI tools, and experience over the course of the study.

## 4.1 Programming Tasks

The eight participants successfully completed the four programming tasks and corresponding questionnaires. As there was no time constraint, participants were given the flexibility to take as much time as they needed to complete the exercises and produce the correct outputs. All submitted code solutions were correct, except for two instances where participants provided a simplified answer that did not accept command-line input but were given a specific input in the code.

Students were permitted to use the internet and any relevant course material, replicating the environment when solving the Inf1B tasks. Six out of the eight participants utilised the internet in at least one task, suggesting that Copilot is not always used as a replacement for online resources or Stack Overflow [31]. Participants' queries varied depending on the student and addressed topics such as: "how to find the maximum in an array", "how to find the index of a value in an array", "reverse function java", "convert integer to string". Most internet searches indicated that the users had a clear understanding of the desired functionality or had prior knowledge in another programming language and were looking for the Java implementation.

The instructions mentioned that Copilot can provide multiple suggestions for a prompt using the `Alt + ]` keyboard shortcut. However, the screen recordings revealed that Copilot rarely offered one or more alternative suggestions. Only three participants used

this feature, inspecting a total of four different suggestions. This was likely due to the introductory quality of the Java tasks and their relatively straightforward solutions.

Furthermore, the screen recordings showed that the coding complexity of the tasks increased over time. Specifically, the average number of minutes spent per task were similar for Tasks 1 and 2 (4.4 and 4.8 minutes, respectively), moderate for Task 4 (6.8 minutes), and significantly higher for Task 3 (15.8 minutes). Task 3, which involved calculating the mode of an array of numbers, proved to confuse many participants, mostly due to their inability to distinguish the maximum entry to the entry with the maximum index.

Task 4 was completed faster than the third task because Copilot recognised the solution when given the name "Sieve of Eratosthenes" in the comments or class name, and immediately suggested the complete solution. This task was deliberately selected to inspect whether students would attempt to code the solution themselves and understand it (also identifying the small changes necessary), or simply accept the suggested solution and proceed to the next task.

### 4.1.1 Queries to Copilot

In this thesis, a query to Copilot is defined as any direct communication or prompt provided to Copilot. Students always issued queries through comments, where they could "instruct" or prompt Copilot. For example, P6 queried: "create a new (array) called copy that has the same numbers but shifted left once". Queries are categorised as Instruction type prompts based on the evaluation criteria outlined in Section 3.5.2. The number of queries made to Copilot per participant and per task (denoted as `no_queries` in Table 3.1) are counted and summarised in Table 4.1.

It is worth noting that not all students felt the need or desire to communicate with Copilot through direct queries, and instead used their code as an indirect prompt. Specifically, only P5, P6 and P7 provided queries to Copilot in every task, while P8 prompted Copilot only once during Task 4.

| | T1 | T2 | T3 | T4 |
|---|---|---|---|---|
| P5 | 8 | 4 | 10 | 4 |
| P6 | 3 | 3 | 7 | 2 |
| P7 | 2 | 2 | 5 | 8 |
| P8 | 0 | 0 | 0 | 1 |

Table 4.1: Queries to Copilot: Participant and Task Counts

There is no clear indication as to the type of participant background that is more likely to make queries to Copilot. Participants P7 and P8 both have advanced programming experience, while P5 and P6 are classified as intermediate-level programmers. However, analysis of the screen recordings exposed two distinct interaction patterns based on the query frequency, which are further analysed in Section 5.1.

### 4.1.2 Measure of acceptance

To quantitatively measure user trust in Copilot and to investigate the relationship between task complexity and acceptance of Copilot's suggestions, we examined the adaptation and acceptance of Copilot's suggestions, a methodology employed in previous studies [41, 25]. Possible measurements included the number of times a user accepted a suggestion from Copilot, as well as whether the accepted code was subsequently edited or deleted. In addition, we counted the instances where a user rejected a Copilot suggestion.

Inspecting Table 4.2, we observe that, on average, there were 64 acceptances per task. Moreover, for Task 3, there were nearly double the number of acceptances compared to the other tasks. As discussed in 4.2, this can be attributed to Task 3 being the most complex among all tasks, leading three students to engage in long debugging sessions. Out of the total number of acceptances, approximately 8% resulted in their subsequent deletion, and 19% of the accepted solutions were later edited by the programmer.

| Task | Accepts | Deletes | Edits | Rejects |
|---|---|---|---|---|
| T1 | 44 | 0 | 3 | 19 |
| T2 | 55 | 5 | 14 | 12 |
| T3 | 99 | 10 | 23 | 18 |
| T4 | 58 | 6 | 9 | 11 |
| Average | 64 | 5 | 12 | 15 |
| Total | 256 | 21 | 49 | 60 |

Table 4.2: Copilot Interaction Summary by Task

### 4.1.3 Prompt Categorisation

To examine how students specify queries to Copilot and how these evolve over time, a prompt categorisation analysis was performed. As shown in Table 4.3, the most common prompt types were the Instruction prompt, Implicit prompt, and Variable declaration prompt, accounting for 25%, 19%, and 16% of all prompts, respectively. The Function declaration type is not as prevalent as reported in the analysis by Perry

| Prompt Type | Count | Prompt % | User % |
|---|---|---|---|
| Function declaration | 14 | 4.9 | 62.5 |
| Method declaration | 40 | 13.9 | 87.5 |
| Variable declaration | 46 | 16.0 | 87.5 |
| None | 54 | 18.8 | 100 |
| Instruction | 71 | 24.7 | 50 |
| Short | 20 | 6.9 | 50 |
| Long | 27 | 9.4 | 37.5 |
| Text close | 15 | 5.2 | 37.5 |
| Library | 1 | 0.3 | 12.5 |

Table 4.3: Prompt Type Distribution and User Engagement

et al. [25], where it was identified as the type that led to stronger participant trust in Copilot's outputs and accounted for 27% of all prompts. This discrepancy likely arises because the tasks in our user study could be solved using a single function without any arguments. Additionally, we observe that long instruction prompts were used more often than short (9% compared to 7%), but more participants preferred to provide shorter, specific prompts to Copilot. Variable declarations were also quite popular as they lead to quick acceptances and accurate predictions when used with a specific variable name. Across all screen recordings, there is only one library prompt (`java.util.*`), as the introductory Java tasks that did not require users to import any libraries. It is important to note that a single prompt may belong to more than one prompt category (e.g. categories Instruction and Long in Table 4.3.)

Interestingly, all participants often accepted suggestions that Copilot provided without an explicit prompt (`none` category). This is due to Copilot's ability to gather data from the entire programming context and to produce suggestions without waiting for a request like other language models. Barke et al. [8] addressed this fact as "unintentional prompting" and defined the respective mode of Copilot as an "Intelligent auto-completion" tool. Nevertheless, sometimes these no-prompt suggestions confused the students and shifted their focus to reading the suggested code instead of writing [31].

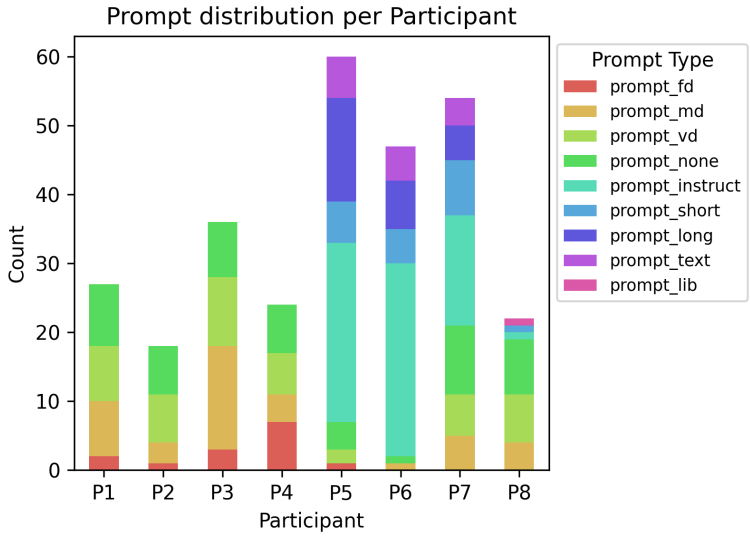


Figure 4.1: Prompt distribution per participant across different prompt types.

Analysing the distribution of prompt types across participants in Figure 4.1, an interesting observation emerges that divides the participants into two categories. First, we observe that P5, P6 and P7 had the highest total count of prompts over the user study, and the most common prompt they used was the Instruction prompt. This contrasts with the rest of the participants, who most often used Variable and Method declaration types. This observation led to the creation of two distinct interaction patterns with Copilot: Collaborator and Operator (refer to Section 5.1).

Task 3 had a significantly higher prompt count than any other task, as shown in Figure 4.2. This was due to the higher complexity of the task and the longer time spent

debugging the code. The rest of the tasks, however, had about the same total count and distribution across prompt types.

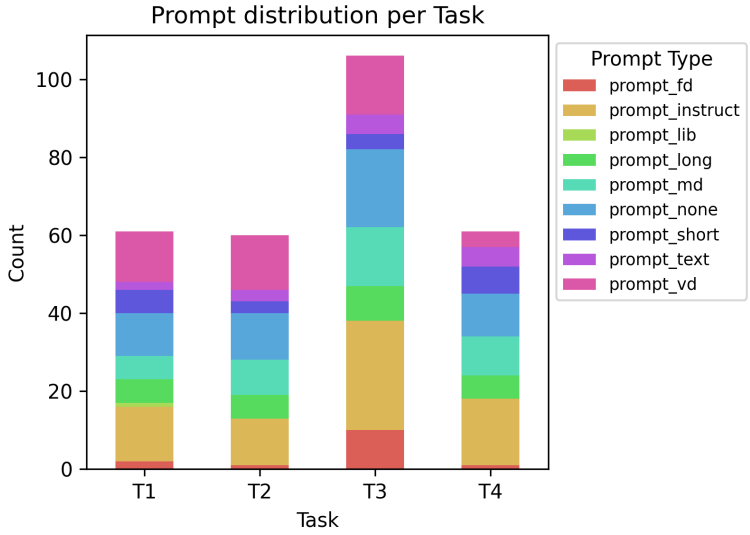


Figure 4.2: Prompt distribution per task across different prompt types.

Therefore, it was not possible to derive a clear conclusion regarding what type of prompts led to stronger acceptance of Copilot's suggestions. The prompting style appeared to depend mainly on the participant. However, through the analysis of screen recordings, we observed that prompt types such as Variable, Method and Function declaration led to faster task completion, as students were primarily guiding the AI tool towards the solution. However, it was clear that Instruction prompts often did not lead to the correct suggestion due to very little or too much detail provided in the instruction.

Additionally, no specific prompt type was identified that led to stronger trust towards Copilot. Participants with advanced programming experience exhibited the lowest level of trust in Copilot, despite having the same prompting style to other participants, such as P3 and P4, who reported high levels of trust in Copilot.

### 4.1.4 Repair Strategies

The analysis of the screen recordings revealed a total of 10 instances of the `continue` repair strategy, 17 instances of `expand`, 7 instances of `reword` and 3 instances of `retry`. Supporting the findings in the study by Perry et al. [25], we found that participants most often expanded the scope of their prompt to Copilot to provide it with more information and improve the quality of suggestions.

However, expanding the prompt through comments proved to be tiresome for participants P5 and P6, as they had to delete the existing code under the prompt every time they added new information and demanded a new suggestion. We also observed a few instances where participants continued typing when they realised that Copilot had not yet understood their approach. In other words, apart from attempting to explain the task

in more detail, some users did not adapt to Copilot's behaviour, but continued with their implementation hoping that Copilot would adapt to their coding approach.

## 4.2 Survey Responses

In this section we present the data collected from the students' survey responses. Three questions were posed after each task to compare the specific screen recording interaction and the response to the question. These addressed topics related to trust, helpfulness and correctness. The remaining questions, which were asked during the first week and at the end of the study, focused on the topics of trust and perceived productivity.

### 4.2.1 Quantitative survey data

We initially analysed the structured questions in Surveys 2 and 3 where students were able to select from a list of possible answers.

#### 4.2.1.1 Per task questions

The first question asked of participants after completing each task was whether they trusted Copilot for their answer. Figure 4.3 shows the distribution of responses for each task. Four participants reported trusting the tool in every interaction. However, three participants oscillated in their responses. The losses of trust occurred for several reasons. Firstly, P1 was confused by a longer suggestion in Task 2, which led them to delete it to avoid potential errors. P1 also did not trust the tool when it suggested an approach to the task that differed from their own. Secondly, P4 only trusted Copilot in the final task because, with a single prompt, they were able to obtain the complete suggested method, which they accepted. Finally, P7 lost trust in Task 3 because Copilot led them to a prolonged debugging session.

Consequently, trust to the AI code-generator is easily lost and highly dependent on the correctness and completion time of the task at hand. Students, especially novice programmers (in this case, due to the Java programming language), easily become confused by the mistakes in the code and quickly question their trust towards Copilot.

Overall, at least 50% of the participants trusted Copilot in every task, with Copilot being more trustworthy in Tasks 1 and 4 (Figure 4.3). This might be due to the fact that Task 1 was the easiest, and Copilot provided correct and accurate answers to all participants, while in Task 4, Copilot knew the answer to the "Sieve of Eratosthenes" problem and could produce the whole function with a single prompt.

The second question posed after the completion of each programming task was to rate the helpfulness of the suggested code by Copilot on a scale from 1 to 5, with 5 indicating very helpful. The purpose of this question was to understand if perceived helpfulness affected the students' trust in Copilot during programming. While there were varying distributions of answers for each question, a ranking of 4 or 5 was the most frequent per task and the average helpfulness ranking per task was consistently 4.

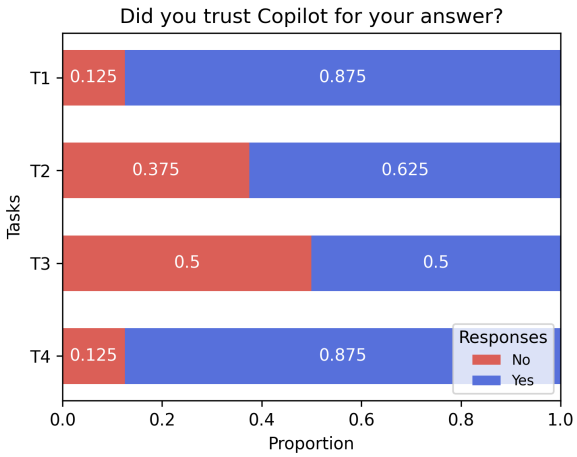


Figure 4.3: Proportion of participants and their level of trust in Copilot per task.

#### 4.2.1.2 Across study questions

We then studied how participants' views of Copilot changed after two weeks of interacting with the tool. We made interesting observations, as for seven out of eight participants, this was their first interaction with Copilot, allowing us to compare results with the paper by Prather et al. [26].

The surveys asked the students to reflect on their overall trust towards Copilot. At the beginning of the study, 50% of the participants reported high trust in Copilot, and 12.5% reported complete trust. After interacting with the tool for two weeks, 62.5% and 12.5% of the students still had high or complete trust, respectively, towards Copilot. For six of the participants there was no change in the overall trust level over the study period, while for P7, the four tasks proved to increase their trust towards the AI. However, trust in Copilot decreased from a moderate to a low trust level after the four tasks for P1.

Furthermore, the surveys asked the participants to state whether their trust in Copilot changed after their weekly interactions. Five participants reported that their trust did indeed change, and based on the high trust percentages reported above, this suggests that the interaction positively affected their trust level. In contrast, two out of the three programmers with advanced experience remained unaffected, reporting no change across the study. Both of these participants had reported moderate or low overall trust levels towards Copilot.

Finally, we posed three final questions shown in Table 4.4 and report their answers. Analysing the responses, we aim to investigate which factors are important in altering users' trust in Copilot over time.

#### 4.2.1.3 Privacy, plagiarism and Copilot

The surveys aimed to gauge how university students, especially undergraduates who must learn proper code citation and referencing, consider code reuse and intellectual property rights when using generative AI tools like Copilot. Copilot's AI model was trained on publicly available code and, in rare instances, its suggestions may

| Question | Response (minutes) | Week 1 (%) | Week 2 (%) |
|---|---|---|---|
| Estimate task completion time without Copilot. | Same | 0 | 0 |
| | 1-3 | 25 | 25 |
| | 5 | 25 | 12.5 |
| | 10 | 37.5 | 0 |
| | $> 10$ | 12.5 | 62.5 |
| Feel more confident coding with Copilot? | Yes | 62.5 | 62.5 |
| | Maybe | 12.5 | 25 |
| | No | 25 | 12.5 |
| Would you trust Copilot for more complex tasks? | Yes | 25 | 12.5 |
| | Maybe | 62.5 | 75 |
| | No | 12.5 | 12.5 |

Table 4.4: Survey responses on Copilot use over two weeks

resemble copyrighted code, raising potential infringement concerns if used without attribution. Copilot provides filters to suppress closely matching public code and a "code referencing" feature to identify relevant open source licenses, although none of the participants utilized these filters [3].

We asked students if they were concerned with plagiarism issues when coding with Copilot during Week 1 and at the end of Week 2. In both instances, only one student answered affirmatively, two students answered "Maybe", while the rest did not express concern about the source of the suggested code.

### 4.2.2 Qualitative survey data

In this thesis, we highly valued open-ended feedback such as comments, ratings and suggestions for feature improvements, to better understand the reasons behind the actions and feelings of the students. Additionally, open-responses provided useful insights that could not be detected through the programming tasks.

#### 4.2.2.1 Trust in AI

Before solving programming tasks, we wanted to understand students' perception of trust in an AI tool. Hence, we asked them to provide us with keywords, examples or a short explanation of what trust in the AI tool means to them. All participants included the topic of accuracy in their response using keywords like "accurately" and"correct answer". This was further confirmed when all participants responded that the accuracy of the AI tool has the greatest impact on building trust between users and AI.

Moreover, half the participants mentioned that a trustworthy tool is able to predict the user's intentions and understand their thinking. Interestingly, P3 connected trust to a "human approach" of the solution. Another common answer was that the AI tool would be able to fix errors in the code and "not lead to any severe problems". Figure 4.5 also confirmed that students expect such tools to find their mistakes, probably since other LLM tools like ChatGPT can point out the errors [37]. However, as discussed in Subsection 4.2.2 most users did not meet their expectation; a very probable reason as to

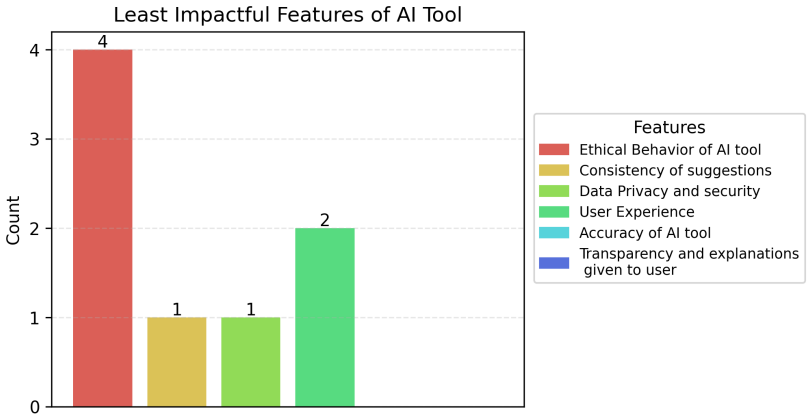


Figure 4.4: Features of the AI tool perceived as least impactful by participants.

why they did not trust Copilot [7].

Finally, one student said that in order to completely trust the AI tool, the user would need to "understand its actions and why it did them". That is an important observation, as transparency is listed as one of the key requirements published by the European Commission for a trustworthy system [13] and has been investigated recently by other papers [34, 32, 14].

The analysis also considered the features of an AI tool that have the less impression on developing trust. Students were given the options listed in Figure 4.4 and asked to select the less impactful feature. Half of the students were least interested in the Ethical behaviour of the AI tool, while two students selected User experience.

#### 4.2.2.2 Initial Expectations from Copilot

The initial expectations of the students towards Copilot are of significant importance to this thesis. By collectively and individually analysing the participants' responses, we investigated whether these expectations influenced their trust in Copilot. In Survey 1 (appendix), participants were asked to select at most three expectations out of nine possible options (including the option "Other"), and their distribution is shown in Figure 4.5. The results indicated that if expectations were met, trust in Copilot increased, whereas if expectations fell short, users' trust was negatively affected.

We observe that all participants expected Copilot to increase their productivity and save time when coding. This was further confirmed in Table 4.4, where all participants predicted an increase in coding time without Copilot's help. As shown in the study by Peng et al. [24] across 95 professional programmers, Copilot met the participants' expectation of increased productivity. For example, P4 commented: "From the tasks I have done so far, I think it will save me a lot of time from defining tedious functions and algorithms".

It is also worth mentioning that the participants new to Copilot expressed an expectation that the tool would find and correct mistakes in their code. However, this expectation was not met, since Copilot is unable to point out the errors in the existing code and to

edit existing mistakes even when requested. Half of the participants mentioned this issue in the open response questions, with P6 suggesting the need for a Copilot debugging tool to address these errors. “Just tell me to correct runtime errors, like index errors, before running” stated P3.

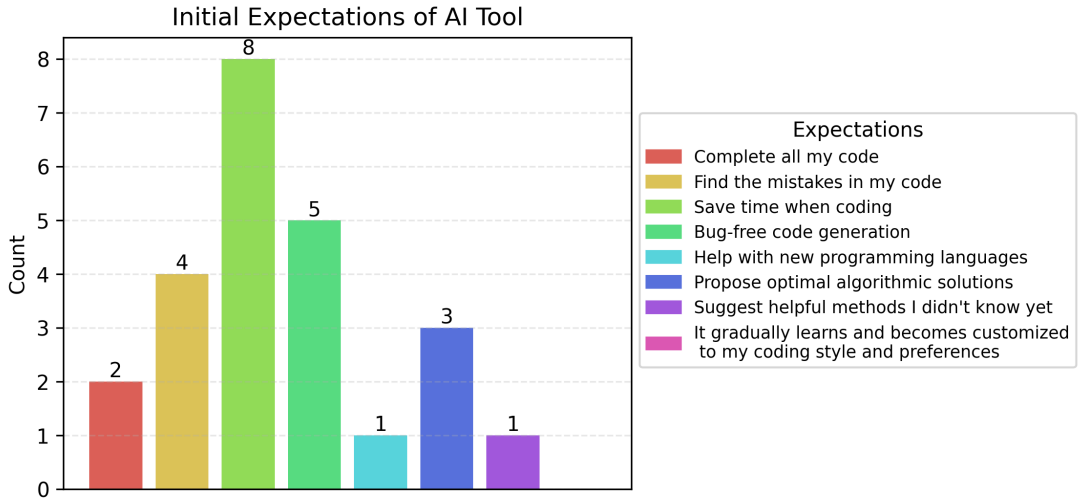


Figure 4.5: Initial expectations of AI tool capabilities from participants.

#### 4.2.2.3 Participants’ Comments on interactions with Copilot

Two questions in Surveys 2 and 3 addressed the beneficial and hindering features of Copilot. A thematic analysis was conducted on the responses, and the prevalent themes were identified and aligned with findings from related studies [31, 8, 26].

The advantages of Copilot addressed by the eight students centered around efficiency and automation, such as improved code quality, increased speed and supplementary learning opportunities. Participants acknowledged Copilot’s ability to automate repetitive and simple tasks and to generate boilerplate code. In other words, the AI tool was perceived as highly efficient in suggesting and auto-completing clean code. Furthermore, users highlighted Copilot’s adherence to good coding practices, as it suggested readable, well-formatted, and syntactically correct code, accompanied by appropriate documentation and comments. P3 described Copilot’s capabilities as “a complete game-changer in my documentation and semi-colon game”. Finally, users noted Copilot’s strength in introducing new, useful methods or code practices that broadened their knowledge.

Conversely, participants also identified several hindering factors associated with Copilot’s usage. A recurring concern was Copilot’s tendency to overwhelm students, either by generating large amounts of code or through its instantaneous suggestions. P2 wrote: “The fact that it suggests immediately without any user input on what the code is meant to do, may sometimes cause the suggestions to be unhelpful”, while P8 also added that these large suggestions tend to assume too much of the problem. Likewise, P4 described that “there were also 1 or 2 times where I began to think about and write a simple statement and Copilot overwhelmed and distracted me by suggesting a massive algorithm”. While Copilot’s instantaneous suggestions were initially impressive, they

sometimes proved tiring and annoying. For instance, P7 said that Copilot "was filling in my comments before I could type my thoughts".

Some usability issues were also highlighted as frustrating. One student reported the difficulty with the multi-key press required for instigating next suggestions, while most users were hesitant or unsure of how to initially use the tool. Additionally, users quickly recognised the need for specificity when creating variable or function names to elicit more detailed suggestions from the AI. Students who relied on comments to instruct Copilot expressed frustration with having to delete existing code suggestions to re-request suggestions when expanding or modifying the scope of their instruction prompt.

#### 4.2.2.4 Common Themes: Task decomposition

A recurrent observation among students was that although Copilot's suggestions were impressive in terms of correctness and speed, they did not always align with the users' intended actions. Hence, students had to guide Copilot towards the desired solution, similar to the "Shepherding" mode described by Prather et al. [26]. P2 provided an insightful explanation: "Copilot worked well when I broke down everything. The more generic my comments, the more difficult it is (for Copilot) to understand what I am asking".

P6 and P2 also noted the necessity of being very specific with the function or variable naming and the comments they provided Copilot. P5 commented that the Inf1B tasks were relatively simple and broken down into easy-to-follow sub-tasks, which facilitated Copilot's understanding. However, for more complex tasks, P5 stated that while Copilot could serve as good starting point, its capabilities were limited to the programmer's understanding. Expanding this point, P1 added that for more complicated tasks, Copilot "requires input from the programmer, often having to override whatever Copilot thinks is the best."

Therefore, Copilot proved to be a useful starting point and a helpful auto-complete tool for straightforward coding tasks. Nevertheless, for more complicated tasks, the responsibility fell on the user to break down the task into smaller structures and guide Copilot towards the correct solution.

#### 4.2.2.5 Common themes: Debugging the suggestion

A prominent theme that emerged from both the screen-recordings and the survey comments regarded the need for debugging Copilot's solution. In numerous instances, the user accepted the incorrect suggestions without initially identifying the errors. When running the program later, users encountered difficulties in resolving the issue, as they had not written and fully understood the suggested code. This was particularly prevalent in Task 3, which required users to calculate the mode of a list of integers.

P7 commented that "In one task I followed Copilot's suggestions because I was unsure as to how to proceed". However, upon realising that the suggestion was incorrect, "it was then really difficult (...) to understand where in Copilot's code was the mistake". P3 encountered a similar situation, stating that they "fell into a kind of panic rabbithole"

in Task 4 and had to consult online resources once they realised Copilot's code was incorrect. As P3 copied methods from the Internet that they did not fully comprehend, "Copilot kind of multiplied the chaos and started to give me cues on the new code I have introduced but actually don't know".

Copilot's inability to find and point out errors in the suggested code can be attributed to the nature of LLMs like itself, which are designed to generate text based on patterns in their training data, but lack the ability to reason about the correctness or functionality of the generated output [39]. Contrary to other LLMs like ChatGPT, Copilot cannot be asked directly about errors or act as a conversational agent.

#### 4.2.2.6 Suggesting Features

The two post-programming task surveys asked the users to indicate if there is a feature they would add or remove from Copilot. This question was added in order to compare our study's responses to the results of Wang et al. [35] , where the researchers conducted a design analysis for the AI tool. We identified three common features in the responses. Firstly, users requested a setting that can reduce the scope of the suggested code and buttons that can specifically ask the tool for a suggestion when needed. Secondly, four participants would like Copilot to have debugging capabilities, as finding incorrect code was a common problem for most users. Finally, P7 said that "sometimes I would like to have an explanation on what the suggested code is doing and why it chose that approach", highlighting the comment by Vaithilingam et al. [31] that Copilot cannot eliminate Internet search, since it cannot provide the explainability of its responses.

# Chapter 5

# Evaluation and Discussion

This chapter presents the evaluation and discussion of the quantitative and qualitative results from our user study. Initially, the data was thoroughly explored from different perspectives to gain familiarity. After examining the survey responses in conjunction with the respective programming task recordings, common themes and interesting comments were identified. The following sections analyse the students' interaction patterns, acceptance metrics and perceived productivity results, and address the issue of over-reliance and the implications of using Copilot in educational settings.

## 5.1 Interaction Patterns: Collaborators vs. Operators

A principal finding of our study was the identification of two distinct interaction styles exhibited by student programmers when using Copilot to solve their class exercises: the *Collaborator*s and the *Operators* types. Collaborators, akin to the "Acceleration mode" defined by Barke et al. [8], possess a clear understanding of their goals and the approach to solve the tasks, and thus collaborate with Copilot to solve the exercise. In contrast, the Operators tend to put significantly less effort in programming and employ Copilot as an operational tool. They provide natural language instructions and rely on Copilot to suggest the appropriate code. These distinct interaction patterns were identified through the prompt categorisation analysis detailed in Subsection 3.5.2 and the screen recording data.

### 5.1.1 Collaborators

Five participants were identified as Collaborators in our user study (P1, P2, P3, P4 and P8). P1 and P8, the most assertive Collaborators, used Copilot as an "intelligent auto-complete", increasing their productivity [10]. They approached tasks with a clear understanding and leveraged Copilot to verify their thinking and complete their thought process. Additionally, they were the only students where Copilot saved them 1-3 minutes of coding (see Table 4.4). Notably, they were hesitant to accept large or complicated code suggestions from Copilot when it "raced ahead", as they did not want to be distracted from their goal. Interestingly, both P1 and P8 relied on the Internet as

their main validation tool, with P8 searching online and copy-pasting a function that returns the `argmax` of an array instead of trying to elicit it from Copilot.

P2, P3, and P4's coding was not as dynamic, as they were less experienced in Java. Nevertheless, they attempted to guide Copilot towards the desired answer and to test their coding knowledge. They spent time validating and understanding the suggested solutions before accepting them, unlike the Operators. Moreover, P2 and P4 run the program immediately after accepting longer Copilot suggestions to identify potential mistakes, a strategy they learned as the study progressed. We noted a shift towards online resources when participants received an erroneous suggestion from Copilot, indicating that trust can be easily lost.

During the Collaboration interaction, participants often edited Copilot's suggestions after accepting them or accepted them line-by-line, sometimes to increase speed, as seen with P3 accepting a for-loop implementation and subsequently editing it for the desired output.

### 5.1.2 Operators

Three participants (P5, P6, and P7), adopted the Operator style of "communication" with Copilot, providing instructions through code comments and writing very few lines of code themselves. The Operators focused on operating Copilot by providing natural language prompts, allowing the AI assistant to guide them towards the solution, define the variable names, and determine the complexity of the approach.

P6 and P7 exemplified this pattern by breaking down the task instructions into short comments for Copilot. For example, to reverse the words in a string, P6 first prompted "break the string down based on the spaces" and then followed with "reverse the order of all the words in the string." Similarly, P7 dissected Task 4 into manageable sub-tasks, such as "use a while loop to check that p $<$ n" and "if not found, set p to n+1".

However, the Operators struggled in identifying and understanding errors in Copilot's suggestions. P6 accepted Copilot's suggestion without validation, proceeding to debug only when the answer provided incorrect. A recurrent issue for the Operators arose when they expanded or reworded their instructions, as re-eliciting a suggestion from Copilot required deleting all subsequent code. Furthermore, some Operators provided complete task instructions in a single, extensive comment before the Java class. Instructions such as "I want this function to be called (...)" adopted a conversational style that may have hindered Copilot's ability to provide accurate suggestions, as it has not been trained on similar natural language prompts in GitHub repositories.

### 5.1.3 Comparison with other interaction styles

Relevant user studies have identified interaction patterns similar to the Collaborator and Operator types, albeit with different focus areas. Prather et al. [26] defined the "Shepherding" and "Drifting" interaction styles. The former involved adapting suggestions to correct distracted workflows, and the latter described instances where large code blocks

led to time-consuming debugging. Our findings confirm the occurrence of “Drifting”, since students were often overwhelmed by Copilot’s extensive code suggestions.

Vaithilingam et al. [31] reported that Copilot often served as a substitute for internet search in their study. While this substitution was observed among our Operators, the Collaborators maintained a higher level of trust in online sources. Furthermore, when Copilot produced incorrect suggestions, not all Operators adopted the coping strategy described in Vaithilingam et al. [31], where participants deleted the suggested code and turned to online help. Instead, some attempted to patch the existing code with online answers, inadvertently compounding the issues.

The “Acceleration mode” defined by Barke et al. [8] was evident in both interaction patterns in our user study. Even passive Operators sometimes decomposed tasks and provided Copilot with short yet descriptive instructions. Our findings also align with the common acceptance of end-of-line suggestions during “Acceleration”, allowing for faster programming. However, we did not frequently observe the smooth transition between the modes described by Vaithilingam et al. [31], nor between Collaborators and Operators.

### 5.1.4 Programming experience vs. interaction patterns

Collaborator and Operator interaction patterns prompted an analysis of the relationship between programming experience, interaction pattern, and trust in Copilot. Figure 5.1 depicts the trust levels of the participants based on their programming experience during the first and second week of the study (left and right plots respectively in Figure 5.1).

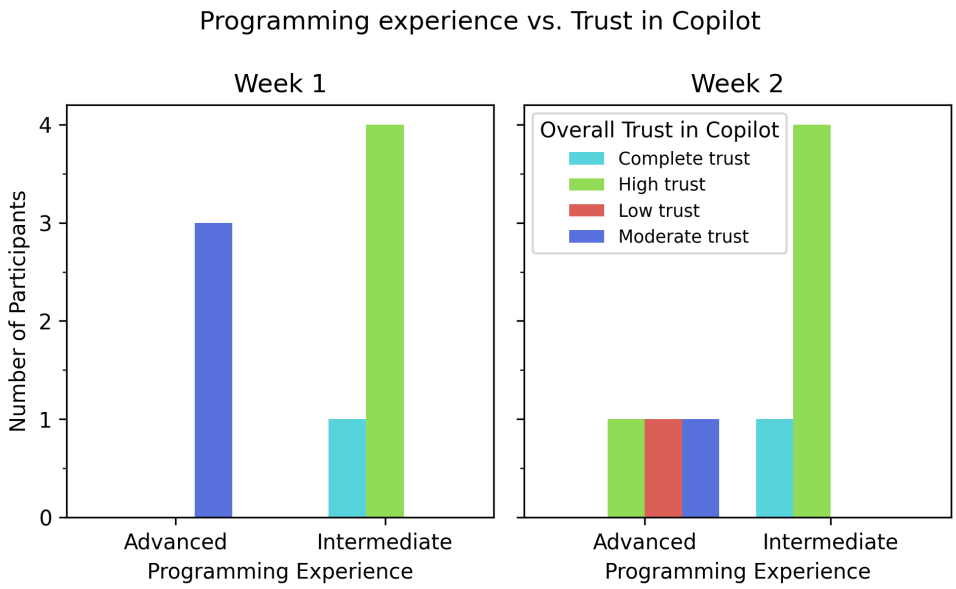


Figure 5.1: Participants’ trust towards Copilot during Weeks 1 and 2, depending on their programming experience level.

After completing Task 1, all advanced programmers exhibited moderate trust towards Copilot, while the intermediate-level students either highly or completely trusted the AI

assistant. However, after the second week's programming tasks, the overall trust levels of the advanced programmers varied from high to moderate and low, whereas for the other participants they remained unchanged.

The participants exhibiting lower levels of overall trust towards Copilot were identified as two users who adopted the Collaborator style. Barke et al. [8] attribute this observation to the excitement and expectations that novice programmers have for AI tools, leading to higher trust and inclination to let it handle the complete coding task.

## 5.2 Copilot and Education

Generative AI code generation tools like Copilot can have both positive and negative impacts on programming education. This study's findings highlight several potential benefits, including improved code quality, better commenting practices, and exposure to varying solutions, which can enhance problem-solving skills and broaden students' experience [9, 17]. Additionally, the screen recordings revealed that many participants, irrespective of their Java programming background, found Copilot relatively easy to use, suggesting that it is a useful tool for novice programmers and learning new material [11, 20]. As noted by Becker et al. [9], positive emotions stemming from successful interactions with Copilot can directly benefit students who may be intimidated or anxious to learn to program.

However, our analysis also identified concerns and potential shortcomings. The two distinct interaction patterns observed (see Section 5.1), showed that students following the Operator programming style, where they adopt a more passive approach to programming and do not proofread the solutions, may hinder their learning [26]. Over-reliance [15, 9], dependence on Copilot's outputs, and high levels of trust and perceived productivity, as evident from our findings, can negatively affect students, especially novices in programming or a specific programming language [15, 8, 26]. Furthermore, high trust and dependence to Copilot may discourage students from attending tutorials, interacting with peers, or seeking explanations from instructors, potentially discouraging their university experience.

[16] Additionally, the use of generative AI tool raises concerns about plagiarism and academic misconduct [9, 16, 26]. Through the survey responses detailed in Section 4.2, we observed that most students were not concerned about possible plagiarism issues when using Copilot. It is crucial for educators and universities to highlight these issues, provide clear guidelines and policies to ensure their ethical and responsible use. Students must exercise due diligence, review applicable licenses, and provide proper attribution when using AI generated code [22].

We consulted the professor and teaching assistants of the Inf1B course about the instructions provided to students regarding the use of generative AI assistants. In summary, Inf1B course allows and even encourages the use of generative AI as a learning tool, but assessments are designed to prevent students from simply submitting AI-generated content without understanding it. Therefore, educators may need to adapt their assessment strategies to motivate students to use the AI tools effectively for their learning.

## 5.3 Acceptance

We conducted an analysis to investigate whether a correlation exists between the acceptance of Copilot's suggestions and the participants' trust level. However, the acceptance counts in our study did not exhibit statistically significant results. Accurately measuring acceptances through screen recordings proved to be an opaque task. While some multi-line suggestions contained the complete code for a function, leading to a single acceptance, others involved multiple shorter single-line acceptances. Despite providing the same overall result, this variation made it challenging to assess trust through acceptance cases and normalise the acceptance counts.

Our findings, however, indicated that the number of acceptances increased with task complexity because students engaged in longer coding session and encountered more debugging scenarios. Task 3 exhibited the highest number of acceptances, while Task 1 had the least. Moreover, we observed that students did not always validate or search for errors in a suggestion before accepting it. For Operators, who followed a passive programming style, suggestions were immediately accepted, and errors were identified only when the code was tested.

## 5.4 Trust and Productivity

Analysis of the survey responses, as shown in Figure 4.5, indicated that all programmers expected Copilot to help them to complete the tasks faster. This observation was particularly significant during the second week of the study, were more than half the participants reported that it would have taken them more than ten minutes to complete the tasks without Copilot's assistance.

The notion of "perceived productivity", as thoroughly analysed by Ziegler et al. [41], is an important topic for students. Our findings suggest that the productivity gains that students experience when coding with AI tools have a significant effect on maintaining their trust towards these tools. This change was exhibited when participants were displeased with Copilot's suggestions and spent time fixing an issue.

University students aim to learn as quickly as possible and need to be highly productive to complete the required course assignments. As highlighted by relevant studies, trust in Copilot is heavily affected by users' expectations of the AI tool's capabilities [35, 24]. Participants in our study demonstrated that even when Copilot produced incorrect suggestions, their perception of increased productivity was enough to motivate them to continue using it.

## 5.5 Over-reliance

A significant concern that emerged from our analysis is the potential for over-reliance on AI coding assistants like Copilot, particularly among novice programmers. Previous research has highlighted this issue, suggesting that inexperienced users may develop a false sense of security, readily trusting the AI's suggestions [25], or even allowing the AI tool to divert them from their initial approaches [15, 8]. The study by Vasconcelos

et al. [32] also showed that AI explanations, such as Copilot's comments, do not always reduce over-reliance but they depend on specific conditions, such as task difficulty.

The high acceptance rates of Copilot's suggestions coupled with relatively low rejection rates suggest that students may have been too readily accepting Copilot's outputs without sufficient scrutiny or validation. Moreover, the increased reliance on Copilot during the complex Task 3, shows that students resorted to accepting and modifying Copilot's suggestions, potentially hindering their problem-solving abilities. This aligns with the report by Passi and Vorvoreanu [22], which states that as the AI tool performed accurately during the initial easier tasks, users tended to over-rely on it for subsequent tasks.

Furthermore, the prevalent use of the Implicit prompt category (18.8% of all prompts), where participants accepted Copilot's suggestions without providing any explicit prompt, could signify over-dependence on the AI assistant's unprompted outputs. Finally, the observation that students had high expectations of Copilot and displayed high levels of trust aligns with the findings by Vodrahalli et al. [33]. Their research concluded that trust in AI systems depends on the perceived performance of the AI relative to humans, with trust increasing when the AI is expected to outperform humans.

# Chapter 6

# Conclusions

This thesis aimed to investigate the interactions between computer science undergraduate students and GitHub Copilot, and to gain insights into the factors influencing user trust. In this chapter we answer the research questions based on our user study's findings, discuss the study's limitations and outline directions for future work.

## 6.1 Research Questions

**RQ1: How do students specify queries to Copilot, and how do these queries evolve over time? Additionally, how do users' interactions with Copilot adapt in response to its behavior?**

We identified two distinct interaction patterns with Copilot that depend on the prompting strategy employed by the students. Five participants exhibited a *Collaborator* style, where they did not specify natural language queries to Copilot through comments. Instead, they had a clear understanding of their problem-solving approach and guided Copilot's suggestions through their coding practices, using the tool as an "intelligent auto-complete" [10]. In contrast, three participants adopted an *Operator* style, following a more passive approach towards coding and specifying natural language instructions to Copilot through comments, similar to observations by Prather et al. [26]. We observe that most Operators did not spend time validating Copilot's suggestions before accepting them and often encountered long debugging sessions.

Our analysis showed that the prompting style depended on the user's interaction pattern. Overall, Instruction type prompts were most prevalent, consistent with the "Specification" prompt type reported by Perry et al. [25], while Method and Variable declaration types were used by the highest percentage of users (refer to 3.5.2.2). Notably, all participants utilised the Implicit prompt type, highlighting Copilot's capability to generate suggestions based on the broader coding context. The most common repair strategy (see 3.5.2.3) was expanding the Instruction prompt (refer to 3.5.2.3), thus providing Copilot with more information [25].

Over the study's duration, participants did not significantly change their prompting strategies. However, we observe that when multiple Copilot suggestions were incorrect,

participants became confused and resorted to seeking solutions online.

**RQ2: How to define and measure user trust, as it may be reflected in students' interactions with Copilot?**

There is limited research on measuring the trust between humans and AI code generations tools due to trust's dynamic and subjective nature [35, 25, 33]. In our study, all students agreed that the Accuracy of the AI tool is the most important factor in building user trust. We measured trust both quantitatively, using the number of acceptances, edits and deletions of Copilot's suggestions, and qualitatively, through structured survey questions and open-response feedback (see Subsection 3.5.2).

We observe that users with advanced programming experience were reluctant to trust the AI tool, and their trust level did not change significantly over time. In contrast, students new to Copilot reported high or complete levels of trust initially, and each interaction positively increased their trust. Barke et al. [8] attribute this potential over-reliance to the excitement and high expectations of novice programmers.

Although the relationship between user trust and acceptance of Copilot suggestions could not be conclusively determined due to small sample size and the need for normalization of the metric, the analysis of programming screen-recordings showed an increase in number of acceptances during the most complex tasks.

**RQ3: How do factors such as task complexity, students' perceived productivity, and their initial expectations of Copilot influence their trust in the tool?**

The higher acceptance counts and the analysis of students' interactions indicated an increased reliance on Copilot during complex tasks, with most students reporting high trust levels towards the AI tool even when it suggested erroneous solutions. Moreover, Collaborators broke down complex tasks into concise, specific sub-tasks, eliciting optimal solutions more frequently [15].

A key observation was that perceived productivity was of utmost importance for the students, as all participants expected Copilot to save them time when coding. Interestingly, students continued to believe that Copilot increased their productivity even when they entered long debugging sessions, as evidenced by survey responses. However, for a few participants, if this expectation of increased productivity was unmet, their trust in Copilot decreased.

Another initial expectation that negatively affected trust levels when unmet was Copilot's inability to generate bug-free code. In some instances, students entered frustrating debugging sessions and expressed the need for a feature in Copilot to identify and correct errors.

## 6.2 Limitations and Future Work

During the user study we encountered several limitations that impeded our work. Firstly, we were only able to recruit a small number of participants due to the nature and time commitment required for the study. Students had to dedicate time during their busy university schedules, screen-record their interaction (a stressful task for novices), and

without compensation. Additionally, we expected a voluntary bias in the participants, as students interested in AI tools and eager to learn about Copilot were more likely to participate.

Furthermore, we did not gain statistically significant insights from the correlation between trust and acceptance metrics because they required normalisation, either using measures like cyclomatic complexity [36], or code length and complexity of suggestions. We also believe that having a second researcher cross-check the evaluation metrics from the screen recordings could improve reliability.

There are many avenues for future work. For instance, we could investigate students' interactions with Copilot over an extended period (like a university semester) to provide insights into even longer-term results. Findings could examine potential learning curves of Copilot, as some users may slowly adapt to using Copilot effectively [25]. Due to the time constraint, we only considered GitHub Copilot as AI assistant. Extending the study to include other generative AI tools, such as AlphaCode [20], could determine whether similar conclusions hold across the different assistants. Moreover, it would be possible to incorporate programming tasks with longer assignments or projects that students are familiar with, replicating different types of assessment in the university [35]. Finally, future work could consider recruiting students with varying levels of prior experience with Copilot to verify whether the identified interaction patterns are still applicable or if new patterns emerge.

## 6.3 Conclusion

This thesis presents a longitudinal comprehensive study on the trust development between undergraduate students and generative AI coding assistants like GitHub Copilot. The identification of two distinct interaction patterns, *Collaborators* and *Operators*, provides valuable insights for designing AI tools and their adoption in educational settings. The user study revealed factors influencing trust, such as met expectations of perceived productivity and accuracy, task complexity and prior programming experience levels. With AI coding tools becoming increasingly prevalent, understanding the human-AI dynamics and prompting strategies will be essential for their effective integration into computer science education.

# Bibliography


[1] ChatGPT. URL https://openai.com/chatgpt.

[2] Github copilot · your ai pair programmer, . URL https://github.com/features/copilot.

[3] Finding public code that matches github copilot suggestions, . URL https://docs.github.com/en/copilot/using-github-copilot/finding-public-code-that-matches-github-copilot-suggestions.

[4] Intellisense in visual studio code. URL https://code.visualstudio.com/docs/editor/intellisense.

[5] Transformer (machine learning model), October 2023. URL https://en.wikipedia.org/w/index.php?title=Transformer_(machine_learning_model)&oldid=1179625108. Page Version ID: 1179625108.

[6] Kabir Ahuja, Harshita Diddee, Rishav Hada, Millicent Ochieng, Krithika Ramesh, Prachi Jain, Akshay Nambi, Tanuja Ganu, Sameer Segal, Maxamed Axmed, Kalika Bali, and Sunayana Sitaram. MEGA: Multilingual Evaluation of Generative AI, October 2023. URL http://arxiv.org/abs/2303.12528. arXiv:2303.12528 [cs].

[7] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 1–13, New York, NY, USA, May 2019. Association for Computing Machinery. ISBN 978-1-4503-5970-2. doi: 10.1145/3290605.3300233. URL https://dl.acm.org/doi/10.1145/3290605.3300233.

[8] Shraddha Barke, Michael B. James, and Nadia Polikarpova. Grounded Copilot: How Programmers Interact with Code-Generating Models, October 2022. URL http://arxiv.org/abs/2206.15000. arXiv:2206.15000 [cs].

[9] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2023, pages 500–506, New York, NY, USA, March 2023. Association

for Computing Machinery. ISBN 978-1-4503-9431-4. doi: 10.1145/3545945.3569759. URL https://dl.acm.org/doi/10.1145/3545945.3569759.

[10] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, July 2020. URL http://arxiv.org/abs/2005.14165. arXiv:2005.14165 [cs].

[11] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code, July 2021. URL http://arxiv.org/abs/2107.03374. arXiv:2107.03374 [cs].

[12] Colin B. Clement, Dawn Drain, Jonathan Timcheck, Alexey Svyatkovskiy, and Neel Sundaresan. PyMT5: multi-mode translation of natural language and Python code with transformers, October 2020. URL http://arxiv.org/abs/2010.03150. arXiv:2010.03150 [cs].

[13] European Commission. Communication: Building Trust in Human Centric Artificial Intelligence. *COM(2019) 168*, April 2019. URL https://digital-strategy.ec.europa.eu/en/library/communication-building-trust-human-centric-artificial-intelligence.

[14] Tyne Crow, Andrew Luxton-Reilly, and Burkhard Wuensche. Intelligent tutoring systems for programming education: a systematic review. In *Proceedings of the 20th Australasian Computing Education Conference*, ACE '18, pages 53–62, New York, NY, USA, January 2018. Association for Computing Machinery. ISBN 978-1-4503-6340-2. doi: 10.1145/3160489.3160492. URL https://dl.acm.org/doi/10.1145/3160489.3160492.

[15] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, Zhen Ming, and Jiang. GitHub Copilot AI pair programmer: Asset or Liability?, April 2023. URL http://arxiv.org/abs/2206.15331. arXiv:2206.15331 [cs].

[16] Nassim Dehouche. Plagiarism in the age of massive Generative Pre-trained

Transformers (GPT-3). *Ethics in Science and Environmental Politics*, 21:17–23, March 2021. ISSN 1611-8014, 1863-5415. doi: 10.3354/esep00195. URL https://www.int-res.com/abstracts/esep/v21/p17-23/.

[17] James Finnie-Ansley, Paul Denny, Brett Becker, Andrew Luxton-Reilly, and James Prather. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. pages 10–19, February 2022. doi: 10.1145/3511861.3511863.

[18] Roberto Gozalo-Brizuela and Eduardo C. Garrido-Merchán. A survey of Generative AI Applications, June 2023. URL http://arxiv.org/abs/2306.02781. arXiv:2306.02781 [cs].

[19] Ellen Jiang, Edwin Toh, Alejandra Molina, Kristen Olson, Claire Kayacik, Aaron Donsbach, Carrie J Cai, and Michael Terry. Discovering the Syntax and Strategies of Natural Language Programming with Generative Language Models. CHI '22, pages 1–19, New York, NY, USA, April 2022. Association for Computing Machinery. ISBN 978-1-4503-9157-3. doi: 10.1145/3491102.3501870. URL https://dl.acm.org/doi/10.1145/3491102.3501870.

[20] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-Level Code Generation with AlphaCode. *Science*, 378(6624):1092–1097, December 2022. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.abq1158. URL http://arxiv.org/abs/2203.07814. arXiv:2203.07814 [cs].

[21] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A Comprehensive Overview of Large Language Models, February 2024. URL http://arxiv.org/abs/2307.06435. arXiv:2307.06435 [cs].

[22] Samir Passi and Mihaela Vorvoreanu. Overreliance on AI Literature Review. 2022. URL https://www.microsoft.com/en-us/research/uploads/prod/2022/06/Aether-Overreliance-on-AI-Review-Final-6.21.22.pdf.

[23] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 754–768, May 2022. doi: 10.1109/SP46214.2022.9833571. URL https://ieeexplore.ieee.org/document/9833571. ISSN: 2375-1207.

[24] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot, February 2023. URL http://arxiv.org/abs/2302.06590. arXiv:2302.06590 [cs].

[25] Neil Perry, Megha Srivastava, Deepak Kumar, and Dan Boneh. Do Users Write

More Insecure Code with AI Assistants?, December 2022. URL http://arxiv.org/abs/2211.03622. arXiv:2211.03622 [cs].

[26] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. "It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *ACM Transactions on Computer-Human Interaction*, 31(1):4:1–4:31, November 2023. ISSN 1073-0516. doi: 10.1145/3617367. URL https://doi.org/10.1145/3617367.

[27] Alec Radford and Karthik Narasimhan. Improving Language Understanding by Generative Pre-Training. 2018. URL https://www.semanticscholar.org/paper/Improving-Language-Understanding-by-Generative-Radford-Narasimhan/cd18800a0fe0b668a1cc19f2ec95b5003d0a5035.

[28] Balagopal Ramdurai and Prasanna Adhithya. The impact, advancements and applications of generative ai. *International Journal of Computer Science and Engineering*, 10(6):1–8, June 2023.

[29] Gustavo Sandoval, Hammond Pearce, Teo Nys, Ramesh Karri, Siddharth Garg, and Brendan Dolan-Gavitt. Lost at C: A User Study on the Security Implications of Large Language Model Code Assistants, February 2023. URL http://arxiv.org/abs/2208.09727. arXiv:2208.09727 [cs].

[30] E. Soloway. Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM*, 29(9):850–858, September 1986. ISSN 0001-0782. doi: 10.1145/6592.6594. URL https://dl.acm.org/doi/10.1145/6592.6594.

[31] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI EA '22, pages 1–7, New York, NY, USA, April 2022. Association for Computing Machinery. ISBN 978-1-4503-9156-6. doi: 10.1145/3491101.3519665. URL https://dl.acm.org/doi/10.1145/3491101.3519665.

[32] Helena Vasconcelos, Matthew Jörke, Madeleine Grunde-McLaughlin, Tobias Gerstenberg, Michael Bernstein, and Ranjay Krishna. Explanations Can Reduce Overreliance on AI Systems During Decision-Making, January 2023. URL http://arxiv.org/abs/2212.06823. arXiv:2212.06823 [cs] version: 2.

[33] Kailas Vodrahalli, Roxana Daneshjou, Tobias Gerstenberg, and James Zou. Do Humans Trust Advice More if it Comes from AI? An Analysis of Human-AI Interactions. In *Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '22, pages 763–777, New York, NY, USA, July 2022. Association for Computing Machinery. ISBN 978-1-4503-9247-1. doi: 10.1145/3514094.3534150. URL https://dl.acm.org/doi/10.1145/3514094.3534150.

[34] Danding Wang, Qian Yang, Ashraf Abdul, and Brian Y. Lim. Designing Theory-Driven User-Centric Explainable AI. In *Proceedings of the 2019 CHI Conference*

*on Human Factors in Computing Systems*, CHI '19, pages 1–15, New York, NY, USA, May 2019. Association for Computing Machinery. ISBN 978-1-4503-5970-2. doi: 10.1145/3290605.3300831. URL https://dl.acm.org/doi/10.1145/3290605.3300831.

[35] Ruotong Wang, Ruijia Cheng, Denae Ford, and Thomas Zimmermann. Investigating and Designing for Trust in AI-powered Code Generation Tools, May 2023. URL http://arxiv.org/abs/2305.11248. arXiv:2305.11248 [cs].

[36] Arthur Henry Watson, Dolores R. Wallace, and Thomas J. McCabe. *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, 1996. ISBN 978-0-16-053381-5. Google-Books-ID: lysRzUZhc2QC.

[37] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT, February 2023. URL http://arxiv.org/abs/2302.11382. arXiv:2302.11382 [cs].

[38] Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate Before Use: Improving Few-Shot Performance of Language Models, June 2021. URL http://arxiv.org/abs/2102.09690. arXiv:2102.09690 [cs].

[39] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A Survey of Large Language Models, November 2023. URL http://arxiv.org/abs/2303.18223. arXiv:2303.18223 [cs].

[40] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large Language Models Are Human-Level Prompt Engineers, March 2023. URL http://arxiv.org/abs/2211.01910. arXiv:2211.01910 [cs].

[41] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. Productivity assessment of neural code completion. MAPS 2022, pages 21–29, New York, NY, USA, June 2022. Association for Computing Machinery. ISBN 978-1-4503-9273-0. doi: 10.1145/3520312.3534864. URL https://dl.acm.org/doi/10.1145/3520312.3534864.

# Appendix A

# User Study Appendix

## A.1 Study Instructions

# Studying the effect of GPT-3 technology (Copilot) on User Trust

# Instructions on Copilot set-up for Java in IntelliJ

Hey there! Thanks again for taking part in my dissertation user study!
I promise that it will be no hassle at all. Instead, you will gain experience with Copilot - a very interesting and useful tool for your degree as a computer scientist.

Moreover, you will not spend any more time than you would normally spend for the INF1B course. This study is designed using already-existing tutorial exercises for the course. The only thing that we ask you for is to record your screen when you are completing the specified exercises, so we can analyse your interaction with Copilot.

For example, we will analyse the acceptance rate of Copilot's suggestions and prompt format. We are **not** analysing how fast you code or using the recording to grade you. Everything will be anonymised.

# What is Copilot?

GitHub Copilot is an innovative programming tool developed by GitHub in collaboration with OpenAI. It serves as an AI-powered code completion and suggestion system designed to enhance the coding experience for developers. Copilot is built on OpenAI's GPT (Generative Pre-trained Transformer) technology, just like ChatGPT, and can generate code snippets based on the context provided by the user. By analysing the patterns and structures in existing code, Copilot can offer intelligent suggestions, significantly accelerating the coding process and helping novice programmers tackle programming problems.

Here's the official GitHub webpage: GitHub Copilot - Your AI pair-programmer

# Getting GitHub Copilot

This file will help you with getting access to GitHub Copilot. The following three steps will take max 10 minutes. You will need a picture of your student card.
If you already have access to Copilot installed in IntelliJ, great, you can skip these steps!

1. Apply to GitHub Global Campus as a student
2. Set-up Copilot in IntelliJ IDEA
3. Quick Copilot tutorial

## 1. Apply to GitHub Global Campus as a student (5 mins)

Please follow steps 1-8 in GitHub Global Campus.

(In Step 6 - Describe how you plan to use GitHub, you could mention the university courses you are taking, projects or university group projects. A sentence should be enough).

> *If your application is approved, you'll receive a confirmation email. Applications are usually processed within a few days, but it may take longer during peak times, such as during the start of a new semester.*

Having access to GitHub Global Campus, apart from Copilot, offers you many additional resources. Check them out in GitHub Student Developer Pack.

## 2. Set-up Copilot in IntelliJ IDEA (5 mins)

Once your application is approved, your task is to install Copilot in IntelliJ, the ODE you will be using for the Java Tutorials and Labs.

Please follow steps 1-9 in Installing the GitHub Copilot plugin in your JetBrains IDE.

## 3. Quick Copilot tutorial

Copilot is super easy to use. When it is activated in your IDE (IntelliJ), it automatically suggests code when you start typing. You can do many things with the generated suggestion. The following table allows you to easily remember them.

| Action | Windows | MacOS | Linux |
|---|---|---|---|
| Accept suggestion | `Tab` | `Tab` | `Tab` |
| See next suggestion | `Alt` + `]` | `Option` + `]` | `Alt` + `]` |
| See previous suggestion | `Alt` + `[` | `Option` + `[` | `Alt` + `[` |
| Reject all suggestions | `Esc` | `Esc` | `Esc` |
| Accept next word | `Control` + → | `Command` + → | `Control` + → |
| Accept next line | `Control` + `Alt` + → | `Command` + `Control` + → | `Control` + `Alt` + → |
| Open tab with multiple suggestions | `Ctrl` + `Enter` and *"Open GitHub Copilot"* | `Command` + `Shift` + `A` and *"Open GitHub Copilot"* | `Control` + `Enter` and *"Open GitHub Copilot"* |

# Week 2 task - Tutorial01

During Week 2, your task is to complete **Task 3 - esrever** in the Exercises section of the tutorial with the help of Copilot. Simply find the Copilot looking-like icon on the bottom right of your IntelliJ screen and select *Enable Completions*. You can use the Internet, StackOverflow or anything you would normally do to solve this task (hopefully not ChatGPT ;) ).

```
Given a string containing a sentence, how would you output the sentence w
ith the order of words in reverse? For example, given ”I like ice cream”
the output should be ”cream ice like I”.

It may help to consider the given string as a character array.
Don't use any methods that you don't know.
```

## To start & stop screen-recording:

| | Windows | MacOS | Linux |
|---|---|---|---|
| Start/Stop recording | `Windows key` + `Alt` + `R` | `Shift` + `Command` + `5` and select second to last icon | `Cntrl` + `Alt` + `Shift` + `R` |

## Sharing the screen-recording:

You can easily find the video of the screen-recording if you look for the Videos folder in Windows, while on Mac the default is the Desktop location. You can now Copy-Paste it in the shared OneDrive folder, the notification of which you received in your university email (if you completed the fist sign-up survey).
If you didn’t complete the sign-up survey, here is the link again:
https://forms.office.com/e/m9ej0SpgvD

It’s very important to answer this quick questionnaire after your Copilot interaction: Copilot - Week 2 Survey

# Week 3 tasks - Lab03

You reached the final task, thank you for sticking with us! This week, the individual Lab exercises present ground for more interesting results than the Tutorial. Hence, when

completing the Labs in your own time, we please ask you to use Copilot's help in the following exercises:

**1. Lab Sheet Week 3 Q3 - ArrayRotate**

Q3 - ArrayRotate

**2. Lab Sheet Week 3 Q6 - Mode**

Q6 - Mode

**3. Lab Sheet Week 3 Q7 - Sieve of Eratosthenes**

Q7 - Sieve

It's very important to answer this quick questionnaire after your Copilot interaction: Copilot - Week 3 Survey

Aaand that's it! Thanks alot! We will share with you the results of the study once completed : )

## A.2 Survey Questions

### A.2.1 Survey 1

**User Study: Studying the effect of GPT-3 technology (Copilot) on User Trust**

Participant Information and Consent form, see Appendix A.1.

*Thank you for agreeing to partake in this study! You are helping us understand more about AI auto-generation tools and human-computer interaction. You will also gain experience with Copilot, a very interesting and helpful tool for your degree!*

*This survey includes a few background questions. It's important to emphasize that any information provided for this study will remain confidential. Your student number is solely used to send you via email the instructions on installing GitHub Copilot.*

1. By choosing "Yes", you consent to take part in this user study and agree with the above statements. (Yes, No)
2. What is your student number? (s1234567)
3. What is your gender? (Female, Male, Non-binary, Prefer not to say)
4. Is English your native language? (Yes, No)
5. What is your overall programming experience level? (Beginner, Intermediate, Advanced)
6. What is your overall programming experience level with Java? (Beginner, Intermediate, Advanced)
7. Have you heard of Copilot before? (Yes, No)
8. If Yes, how often do you use GitHub Copilot? (Daily, Weekly, Monthly, Never)
9. What does "trust in an AI tool" mean to you? Type a few keywords, examples or a short explanation.
10. Would you say that you can trust Copilot?
11. In your opinion, what has the greatest impact on building trust between users and AI?
    (a) Accuracy of AI tool
    (b) Transparency and explanations given to user
    (c) Ethical Behavior of AI tool
    (d) Consistency of suggestions
    (e) Data Privacy and security
    (f) User Experience
    (g) Other

12. In your opinion, what has the least impact on building trust between users and AI? (Same response options as Question 11)
13. What are your expectations of such an AI tool like Copilot? Select at most three.
    (a) Complete all my code
    (b) Find the mistakes in my code
    (c) Save time when coding
    (d) Bug-free code generation
    (e) Help with new programming languages
    (f) Propose optimal algorithmic solutions
    (g) Suggest helpful methods I didn't know yet
    (h) It gradually learns and becomes customized to my coding style and preferences
    (i) Other

### A.2.2 Survey 2

**Copilot - Week 2 Survey** *Please answer this questionnaire after completing the Tutorial task with Copilot. Thank you for completing the week 2 task! You were probably impressed with Copilot's capabilities. Here's a few quick questions on your experience.*

1. What is your student number? (s1234567)
2. Did you trust Copilot for your answer? (Yes, No)
3. Rate the helpfulness of the suggested code by Copilot (1-5)
4. I think I solved this task correctly (Likert 1-5)
5. I trusted the AI to produce correct code (Likert 1-5)
6. What is your overall trust in Copilot?
    (a) Complete trust
    (b) High trust
    (c) Moderate trust
    (d) Low trust
    (e) No trust
7. Did your trust for Copilot change after this interaction? (Yes, No, Maybe)
8. Estimate how much slower you would have completed the task without using Copilot.
    (a) Same time in both cases

   (b) 1-3 minutes slower

   (c) 5 minutes slower

   (d) 10 minutes slower

   (e) More than 10 minutes slower

9. What was annoying while interacting with Copilot?
10. What was helpful while interacting with Copilot?
11. Do you feel more confident when coding with Copilot? (Yes, No, Maybe)
12. Is there a feature you would add/remove from Copilot?
13. Do you worry about the source of the suggested code and any potential issues related to plagiarism? (Yes, No, Maybe)
14. Would you trust Copilot for more complex tasks? (Yes, No, Maybe)
15. Comments?

### A.2.3 Survey 3

**Copilot - Week 3 Survey** *Please answer this questionnaire after completing the Labs tasks with Copilot. Thank you for completing the week 3 tasks! You were probably impressed with Copilot's capabilities. Here's a few quick questions on your experience.*

1. What is your student number? (s1234567)
2. Q3 ArrayRotate: Did you trust Copilot for your answer? (Yes, No)
3. Q3 ArrayRotate: Rate the helpfulness of the suggested code by Copilot. (1-5)
4. Q3 ArrayRotate: I think I solved this task correctly (Likert 1-5)
5. Q3 ArrayRotate: I trusted the AI to produce correct code (Likert 1-5)
6. Q6 Mode: Did you trust Copilot for your answer? (Yes, No)
7. Q6 Mode: Rate the helpfulness of the suggested code by Copilot. (1-5)
8. Q6 Mode: I think I solved this task correctly (Likert 1-5)
9. Q6 Mode: I trusted the AI to produce correct code (Likert 1-5)
10. Q7 Sieve of Eratosthenes: Did you trust Copilot for your answer? (Yes, No)
11. Q7 Sieve of Eratosthenes: Rate the helpfulness of the suggested code by Copilot. (1-5)
12. Q7 Sieve of Eratosthenes: I think I solved this task correctly (Likert 1-5)
13. Q7 Sieve of Eratosthenes: I trusted the AI to produce correct code (Likert 1-5)
14. What is your overall trust in Copilot?

   (a) Complete trust

   (b) High trust

   (c) Moderate trust

   (d) Low trust

   (e) No trust

15. Did your trust for Copilot change after this interaction? (Yes, No, Maybe)
16. Estimate how much slower you would have completed the task without using Copilot.

   (a) Same time in both cases

   (b) 1-3 minutes slower

   (c) 5 minutes slower

   (d) 10 minutes slower

   (e) More than 10 minutes slower

17. What was annoying while interacting with Copilot?
18. What was helpful while interacting with Copilot?
19. Do you feel more confident when coding with Copilot? (Yes, No, Maybe)
20. Is there a feature you would add/remove from Copilot?
21. Do you worry about the source of the suggested code and any potential issues related to plagiarism? (Yes, No, Maybe)
22. Would you trust Copilot for more complex tasks? (Yes, No, Maybe)
23. Comments?

## A.3 Demographics

This table presents the demographics collected from the study's participants during the first survey.

| Demographic | Cohort | Participants | Total % |
|---|---|---|---|
| Gender | Female | 2 | 25 |
| | Male | 6 | 75 |
| | Non-binary | 0 | 0 |
| | Prefer not to say | 0 | 0 |
| Native language | English | 3 | 37.5 |
| | Other | 5 | 62.5 |
| Programming proficiency | Beginner | 0 | 0 |
| | Intermediate | 5 | 62.5 |
| | Advanced | 3 | 37.5 |
| Java proficiency | Beginner | 5 | 62.5 |
| | Intermediate | 3 | 37.5 |
| | Advanced | 0 | 0 |
| Copilot Awareness | Yes | 7 | 87.5 |
| | No | 1 | 12.5 |
| Copilot usage frequency | Daily | 1 | 12.5 |
| | Weekly | 0 | 0 |
| | Monthly | 0 | 0 |
| | Never | 7 | 87.5 |
| Trust in Copilot | Yes | 1 | 12.5 |
| | No | 0 | 0 |
| | Unsure | 4 | 50 |
| | Depends | 3 | 37.5 |

Table A.1: Participant demographics and cohort percentages.

# Appendix B

# Participants' information sheet and consent form

# Participant Information Sheet

| | |
|---|---|
| Project title: | Studying the effect of GPT-3 technology (Copilot) on User Trust |
| Principal investigator: | Dr Kobi Gal |
| Researcher collecting data: | Nephele Aesopou |

This study was certified according to the Informatics Research Ethics Process, reference number 676940. Please take time to read the following information carefully. You should keep this page for your records.

**Who are the researchers?**

This is an UG4 dissertation project led by Nephele Aesopou, Computer Science and Mathematics student at the University of Edinburgh. The project is supervised by Dr Kobi Gal, Reader in Artificial Intelligence and Human-Machine Intelligence at the University of Edinburgh.

**What is the purpose of the study?**

The purpose of the project is to study the effect of Copilot use on user adoption and trust by investigating the interactions between programmers and Copilot and analysing their perceptions on user trust. Specifically, the study will analyse the format of prompts to Copilot, how users adapt to Copilot's behaviour and ways to measure user trust.

**Why have I been asked to take part?**

You have been asked to take part because you fall within the specific user segment it seeks to analyse, University of Edinburgh students with an interest in AI code generation tools and Copilot.

**Do I have to take part?**

No – participation in this study is entirely up to you. You can withdraw from the study at any time, without giving a reason. Your rights will not be affected. If you wish to withdraw, contact the PI. We will stop using your data in any publications or

presentations submitted after you have withdrawn consent. However, we will keep copies of your original consent, and of your withdrawal request.

**What will happen if I decide to take part?**

If you do decide to take part, you will be introduced to the very interesting OpenAI's pair-programmer tool, Copilot. You will be asked to answer some questions in a survey regarding your year of study, age, gender, experience with programming and Copilot and your opinions on user trust in AI technology and Copilot. Then, you will be asked to complete a few coding tasks in Java with the help of Copilot, that will take maximum 20 minutes. You will be asked to record your screen so we can investigate the interaction. You can complete the tasks at your own time, on your personal computer, as long as it is done during the period specified. Do not worry, the correctness or efficiency of your code will not be associated to you or your performance at the university courses; it will immediately be anonymised. After each interaction you will be prompted to answer a survey, with questions reflecting on your interaction with Copilot. Contingent upon your consent, we will ask you to repeat this process again for another tutorial.

**Are there any risks associated with taking part?**

There are no significant risks associated with participation.

**Are there any benefits associated with taking part?**

By taking part in the study, you will gain experience with Java and you will learn how to code using Copilot effectively. Moreover, we will share with you the results of the study.

**What will happen to the results of this study?**

The results of this study may be summarised in published articles, reports and presentations. Quotes or key findings will be anonymized: We will remove any information that could, in our assessment, allow anyone to identify you. With your consent, information can also be used for future research. Your data may be archived for a minimum of 2 year.

**Data protection and confidentiality.**

Your data will be processed in accordance with Data Protection Law. All information collected about you will be kept strictly confidential. Your data will be referred to by a unique participant number rather than by name. Your data will only be viewed by the researcher/research team; Nephele Aesopou and Dr Kobi Gal.

All electronic data will be stored on a password-protected encrypted computer, on the School of Informatics' secure file servers, or on the University's secure encrypted cloud storage services (DataShare, ownCloud, or Sharepoint) and all paper records will be stored in a locked filing cabinet in the PI's office. Your consent information will be kept separately from your responses in order to minimise risk.

**What are my data protection rights?**

The University of Edinburgh is a Data Controller for the information you provide. You have the right to access information held about you. Your right of access can be exercised in accordance Data Protection Law. You also have other rights including rights of correction, erasure and objection. For more details, including the right to lodge a complaint with the Information Commissioner's Office, please visit www.ico.org.uk. Questions, comments and requests about your personal data can also be sent to the University Data Protection Officer at dpo@ed.ac.uk.
For general information about how we use your data, go to: edin.ac/privacy-research

**Who can I contact?**

If you have any further questions about the study, please contact the lead researcher, Nephele Aesopou, by email to s2056170@ed.ac.uk.
If you wish to make a complaint about the study, please contact inf-ethics@inf.ed.ac.uk. When you contact us, please provide the study title and detail the nature of your complaint.

**Updated information.**

If the research project changes in any way, an updated Participant Information Sheet will be made available on http://web.inf.ed.ac.uk/infweb/research/study-updates.

**Consent**

By proceeding with the study, I agree to all of the following statements:

- I have read and understood the above information.
- I understand that my participation is voluntary, and I can withdraw at any time.
- I consent to my anonymised data being used in academic publications and presentations.
- I allow my data to be used in future ethically approved research.

  [Button here named “I agree” or “take me to the survey”]