UNIVERSITÄT
LEIPZIG

Faculty of Mathematics and Computer Science

The Impact of LLM-Based Coding Assistants on Developer Productivity

Master's Thesis

Annemarie Wittig
Matriculation Number 3789345
Born Oct. 21, 1998 in Halle (Saale)

Referee:
1. Prof. Dr. Ing. Norbert Siegmund
2. Prof. Dr. Martin Middendorf

Submission date:
July 30, 2025

Declaration of Independence

I certify that I have completed this thesis independently and only using the specified sources and resources. In particular, any literal or analogous quotations are marked as such. I am aware that violations may result in the subsequent revocation of my degree.
I certify that the electronic copy corresponds to the printed copies.

Leipzig, July 30, 2025

. . . . . .
Annemarie Wittig

Abstract

LLM-based coding assistants have seen widespread adoption in software engineering, yet their actual impact on developer productivity remains uncertain. While surveys and controlled experiments offer some insights, they often yield conflicting results and rely on varying definitions of productivity, limiting comparability across studies. Moreover, few works have empirically linked AI tool usage to changes in the actual output of software development – namely, code.

This thesis addresses that gap by empirically analyzing the effects of LLM-based coding assistants on objective, repository-level productivity metrics. We analyzed data from 14 developers working across 39 repositories, covering periods before and after the introduction of an LLM-based coding assistant into their company's development workflows. Additionally, we examined 6 open-source repositories around the public release of GitHub Copilot. Across both contexts, we applied a suite of 14 privacy-preserving productivity metrics derived from repository data, allowing for a longitudinal and comparative assessment of productivity trends.

Using these findings, we contribute a set of 14 privacy-preserving productivity metrics, 7 validated hypotheses on AI tool impact grounded in real-world data, and a foundation for robust, context-aware productivity research of AI-assisted development. In doing so, we offer a concrete step toward understanding the real-world effects of AI-assisted development using objective, scalable, and privacy-conscious measures.

Contents

Acknowledgments

I want to thank everyone who supported me in writing this thesis.

First and foremost, I would like to thank my supervisor, Norbert Siegmund, for his invaluable guidance throughout this challenging thesis and study. Without his input and encouragement, this work would not be where it is today. Beyond this thesis, I am especially grateful for his help in identifying a topic I truly enjoy, reigniting my passion for science, and setting me on the path toward a doctorate.

I am also grateful to my colleagues and friends, especially Alina Mailach and Max Weber, for their insightful feedback and constant support.

Special thanks to Andreas Both, whose courses during my Bachelor's studies sparked my joy in Computer Science, and with whom I took my first steps into scientific research

To my best friends, Natalie and Leana – thank you for making the writing process enjoyable and keeping spirits high, even during those last few rainy days. And to all my other friends and fellow coworking thesis writers – I genuinely appreciated the quiet writing sessions and the Spaßgetränke breaks.

I deeply appreciate my siblings, each of whom has contributed to my journey in their own way. Especially my two bigger brothers, Richard, who shared with me a passion for gaming that sparked my interest in computer science, and Arthur, a constant source of inspiration and motivation. A heartfelt thank you to my parents: My papa and mama, who have always been there whenever I needed something and helped shape the person I am today. To my grandmother, who has never once stopped telling me how proud she is of me and my achievements. And to the grandmothers I lost, you are dearly missed and I would have loved to share this milestone with you. I also want to thank Peppi, who, despite all the personal challenges this year, never fails to bring a smile to everyone's face.

Finally, to my love, Boris – thank you for your unwavering belief in me, even when I couldn't believe in myself. You've been by my side every step of the way, and I look forward to continuing our journey together.

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Meaning
abbr.	abbreviation
agg.	aggregation
AI	artificial intelligence
CCP	Corrective Commit Probability
DSGVO	Datenschutzgrundverordnung
e.g.	*exempli gratia* – for example
et al.	*et alii* – and others
etc.	*et cetera* – and other similar things
i.e.	*id est* – that is
IR	industry repository – a private repository of one of the companies analyzed in the study
LLM	large language model
LoC	lines of code
OSR	open-source repository
PR	pull request
rel.	relative

Chapter 1

Introduction

Are Coders' Jobs At Risk? AI's Impact On The Future Of Programming
Forbes, Duranton, 2024

AI Coding Assistants Wave Goodbye to Junior Developers
CIO, Ross, 2024

AI Coding Assistants Can Be a Huge Help – Just Not Where You Think
The Guardian, Dhar, 2025

Now you don't even need code to be a programmer. But you do still need expertise
Built In, Naughton, 2025

AI Coding Assistants Are Reshaping Engineering – Not Replacing Engineers
The New Stack, Gootee, 2025

How AI-Assisted Coding Will Change Software Engineering: Hard Truths
The Pragmatic Engineer, Orosz and Osmani, 2025

These newspaper headlines make one fact abundantly clear: LLM-based coding assistants are at an all-time hype, but whether they are actually beneficial to developers remains an open question. Companies are increasingly shifting their focus to integrate more AI into their applications [51] and the majority of professional developers are using, or plan to use, AI in their development process [46]. LLM-based coding assistants are here – and they are here to stay.

These recent trends are driven by the advancements of artificial intelligence (AI), which have led to a new type of generative model: the large language model (LLM). Its ability to process and generate natural language made it interesting for a widespread audience, especially for software developers. The release of OpenAI's Codex [9] in 2021 marked the introduction of the first

widely used LLM-based coding assistant: Github CoPilot[1]. With it, the software engineering community has found itself several new topics of interest, one of which asks the question: How do LLM-based coding assistants (also referred to as AI tools in this thesis) influence professional software developers' productivity?

There have been a multitude of surveys [5, 12, 18, 31, 39, 46, 51, 65, 67, 70], experiments [7, 13, 48, 59, 61, 67], and other research [3, 8, 29] focused on answering this question. However, the results present an unclear and mixed picture of the actual impact. After the introduction of AI tools, the perceived and measured productivity ranges from worse [40], to unchanged [31, 47], to strongly increased [18, 23, 29, 31, 46, 47, 51, 69, 70]. Some studies even report both unchanged and improved productivity within the same sample [31, 47].

The issue begins with the term *productivity* itself, which is defined and measured inconsistently across different studies. Some use different metrics to capture the same underlying concept (e.g., developer's speed measured by the number of commits [2], perceived or measured time to solve a task [5, 13, 31, 47, 59, 65] or learn a new skill [46, 61]), while others target entirely different concepts (e.g., cognitive load [46, 70]). These differences make comparisons between studies, their interpretations, and their insights even more difficult. The research community has not reached a consensus – neither on metric, nor the impact on developer productivity. With results differing depending on the study, and many of the prominent studies being conducted in-house [47, 69, 70] (with potential conflicts of interests), said consensus is unlikely to be reached anytime soon.

Given these developments, it seems natural to shift the focus from developer perceptions and small-scale experiments to the actual code developers produce, which is the product of their work. And while it seems that the overall tendency of developer productivity studies points to positive outcomes, recent reports continue to raise doubts. GitClear [24], drawing on one of the most comprehensive code analyses to date (211M lines of code from 2020–2024), highlights several negative effects of AI tools, including higher defect rates and reduced code reuse and refactoring. Yet, to the best of our knowledge, studies evaluating code specifically in the context of AI tool-induced changes, such as the GitClear report, remain rare. A potential reason are the persistent challenges with code-based productivity metrics.

This scarcity does not reflect a general lack of interest in analyzing productivity through large codebases, or rather, the repositories that contain them. In fact, there is a growing body of research analyzing repository-based productivity [2, 28, 34, 35, 37, 44, 52, 55], just without looking at the impact of

[1] https://github.com/features/copilot

AI tools. Consequently, although there is strong interest in understanding the effects of AI tools on software development and a parallel interest in analyzing productivity through code repositories, the combination of these two perspectives remains underexplored. As of now, we know little about how AI tools affect the very core of software development: The day-to-day code developers actually produce in practice.

With this thesis, we aim to fill the gap between existing research on AI-assisted productivity and repository-based productivity analysis by empirically linking AI-assisted development to measurable changes in source code. Amid mixed research findings, the need to define productivity, and the uncertainty of AI tools' impact on code, our study takes a decisive step toward understanding the real-world impact of LLM-based coding assistants.

To contribute to this goal, we analyze the effect of LLM-based coding assistants on repository productivity using objective, repository-level metrics. We conducted an exploratory study with confirmatory elements in three steps: We **(i)** analyzed six open-source repositories (OSRs) over 990 days (2.7 years) before and after the introduction of GitHub Copilot, **(ii)** performed the same analysis on developers of real-world industry repositories (IRs) developed and maintained within companies for their customers, over a smaller timeframe, compared our findings across both settings, and **(iii)** conducted a survey with professional developers from the same companies to validate, interpret, and correlate the observed effects.

In a nutshell, our research methodology is as follows: We carried out a two part study with three companies, one of which is evaluated in this thesis, involving a total of 27 developers. First, we had the developers complete a survey about their expectations regarding AI tool usage. Then, we collected data from 42 repositories these developers worked on. Using a subset of this data, we computed 14 abstract repository-level metrics that do not require code content analysis, in alignment with the companies' privacy policies. The survey responses served to interpret, validate, and contextualize the observed effects. Second, to strengthen and validate our findings within the companies and to increase external validity, we replicated a similar analysis on a set of 6 OSRs. In research, OSRs are commonly used due to their accessibility and scale. This comparison enables us to assess whether the patterns observed in smaller-scale, company-specific contexts align with those found in larger, publicly available projects.

For the analysis, we first identified which metrics are suitable and in compliance with individual company constraints, such as privacy concerns. Based on related work, we then formulated a set of hypotheses. Beyond testing these, we conducted an exploratory analysis of the metric outcomes, which enabled us to derive additional hypotheses on the impact of AI tools in software devel-

opment.

To summarize, this study empirically evaluates the impact of LLM-based coding assistants on developer productivity by combining objective code output metrics with in-context developer surveys. Our contributions are:

1. A comprehensive suite of privacy-preserving, repository-level productivity metrics, validated across multiple contexts and settings.

2. A set of tested and newly derived hypotheses on the impact of AI tools on repository productivity, based on a data-driven analysis of actual development activity rather than self-reported perceptions or synthetic tasks.

3. A foundation for future research toward methodologically robust and context-aware productivity evaluation in AI-assisted software development.

In this thesis, we begin by discussing the theoretical context of what productivity means, how we define it, what research has been done on the topic, and introduce any other specific terminology used in this work in chapter 2. In chapter 3, we present our methodology, starting with our research questions, their relevance, and how they are operationalized. We then describe our process for computing productivity metrics and the steps taken to interpret them in section 3.2, followed by detailed explanation of each metric, the work they are based on, their rationale, and their formulas. Next, we explain how we grouped and analyzed the questionnaire responses in section 3.3, and present our hypotheses, their rationale, and the corresponding metrics used for testing in section 3.4. We outline the participating companies and study setting in section 3.5. The methodology chapter concludes in section 3.6 with the study materials, including the questionnaires, information about the IRs and OSRs, the collected data, and the tools used for data collection. In chapter 4, we present the results for each research question individually, whereas we interpret them in chapter 5, organized by our hypotheses and exploratory findings. We further discuss potential threats to the validity in chapter 6. We close this thesis with an outlook on potential future extensions, lessons learned from conducting this study, and a final conclusion in chapters 7, 8, and 9.

Chapter 2

On Productivity in Software Engineering

In this chapter, we provide the background for understanding developer productivity in software engineering, focusing on both the use of AI tools and general productivity concepts. We begin by categorizing the productivity term, followed by using the SPACE framework to define what productivity means in this context. Then, we discuss (i) specific productivity effects observed when developers use AI tools and (ii) productivity measures employed in repository-based analyses as previously found in related work.

2.1 What is *Productivity*?

In software engineering research, productivity has been defined countless times, with metrics changing depending on the setting, the context and the purpose of the research. Research continuously finds new aspects and new perspectives on what shows developers' productivity, with some more common dimensions including development speed [5, 13, 29, 31, 46, 47, 59, 61, 65], the quality of code [29, 39, 51, 70], or the developer's perceived impact [18, 23, 29, 46, 51, 69, 70].

In this work, we will differentiate between two general types of productivity. First, the *individual* or *perceived* productivity focuses on developers' experienced and self-reported effects when using AI tools. This can be measured using surveys, interviews, or other study designs that directly involve developers in the data collection process (e.g., sources presented in subsection 2.2.1). Second, the *objective* or *repository-based* productivity combines empirical measurements using objective repository data to draw conclusions about productivity, without directly addressing developers' subjective experi-

ences (e.g., sources presented in subsection 2.2.2).

Research has shown that productivity needs to be viewed through different lenses [30]. As such, we further differentiate between two perspectives: (i) *Developer productivity*, which integrates both individual and objective measures to assess the impact of the AI tools on an individual developer's productivity, i.e., a per-person perspective, and (ii) *Repository productivity*, which focuses solely on objective metrics to evaluate productivity at the level of a project's entire codebase stored in a repository. This perspective explicitly works at the level of the repository and does not focus on analyzing actual code content.

To design a structured approach to measuring productivity, particularly in developing our questionnaire on self-reported impacts and in contextualizing our objective metrics, we adopt the SPACE framework [21] as a guiding concept. Although we could not apply the framework rigorously due to the requirements and constraints of each participating company and us focusing on *objective* measures, it informed our study design and remains relevant for interpreting productivity in software engineering research. Forsgren et al. [21] introduced a framework to measure developer productivity using five dimensions: **S**atisfaction and well-being, **P**erformance, **A**ctivity, **C**ommunication and collaboration, and **E**fficiency and flow. We combine repository-based metrics with a developer survey to cover all dimensions. However, as the questionnaire focuses on expectations, it does not yet allow for a deterministic assessment of developers' actual experiences with the AI tool.

Satisfaction and well-being describes developers' fulfillment with their work, their health and overall happiness. It is inherently subjective to the individual and as such is exclusively covered in the survey. Performance describes the outcome of a system or process, for instance through quality or system health. It is generally hard to assess, and even harder in our case where we cannot analyze actual code. As such, we can only indirectly approximate it through repository data and developers' self-reports. Activity, defined as the volume of developer actions (e.g., the number of commits or pull requests), is not part of our survey but is captured entirely through the objective repository metrics. Communication and collaboration focuses on the interactions within and across teams and developers. It is analyzed from both perspectives, through developer impressions and using repository-based measures, which we describe for entire repositories, thus reflecting collaborative activity rather than individual behavior. Finally, efficiency and flow, which concerns how much progress can be made by developers without being interrupted, are similarly addressed using both self-reported perceptions and repository data reflecting work patterns.

2.2 Related Work

Measuring productivity has been the focus of many researchers, particularly with the rise of LLM-based coding assistants in the software engineering industry. In this section, we discuss related work from two perspectives. First, we present studies examining the impact of AI tools on professional software developers' productivity, from their perceptions and their code-related behavior. Second, we review studies that explore how productivity can be measured using software repositories in general, regardless of the usage of AI tools. Specifically, we focus on works employing abstract, quantifiable metrics that do not rely on the exact content of the code. Only such measures are applicable in the context of our own study within software engineering companies and in compliance with their data privacy policies.

2.2.1 Productivity with AI tools

A key dimension of productivity is its evaluation from the individual developer's perspective. This section reviews prior work on how developers use AI tools, their performance in experiments, and their perceived productivity. We focus on reported findings rather than study methodologies and exclude studies without human subjects or those conducted solely in academic settings.

Most of the studies reviewed rely on surveys with large numbers of developers, either through online recruitment [5, 18, 31, 46, 51, 65, 70] or within specific companies [12, 39, 67]. Approaches more aligned with ours include workplace observations [3, 29], interviews [23], experiments with open-source developers, using actual issues in their software [40], and large-scale multi-week company-based experiments [8]. Another common method involves controlled experiments [7, 13, 48, 59, 61, 67], where participants are assigned tasks to complete with AI tools, allowing researchers to analyze outcomes and oftentimes compare them against outcomes of participants without AI support. These study designs fundamentally differ from ours: rather than targeting broad and undefined developer populations in short experimental settings, we examine developers' coding behavior within multiple companies, while they follow their real-world development activities. This allows us to observe their day-to-day work in context and analyze their behavior within code repositories over long periods of time instead of just getting a momentary snapshot. Due to the structure of the studies discussed in this section, many of the found effects are based on self-reported measurements.

Multiple studies have found developer perceived productivity to be impacted using LLM-based coding assistance, though to varying degrees: some find a strong increase of productivity [18, 23, 29, 46, 51, 69, 70], some find

a minor increase [39, 48, 65, 67], and some find none at all [31]. Sometimes, researchers even found all of these groups represented within one questionnaire [31, 47]. Interestingly, the perceived productivity increases were not always confirmed through statistical analysis of objective productivity measures, which sometimes showed no change [69] – and sometimes did [70]. In some cases, researchers found the impact to be greater for, or even restricted to, less experienced developers [18, 23, 29, 70].

When looking at more specific productivity measures, the speed at which developers created code, solved tasks, or learned new concepts was increased according to most research [5, 13, 29, 31, 46, 47, 59, 61, 65]. The actual speed improvements, while not always defined specifically, range from 30 % to 70 % faster task completion rates, with some studies even reporting close to no, or lesser effects the more complex the measured task is [13, 46]. However, one of the newest studies on development speed, measured by the time to solve issues in the developers' open-source project, contradicts these findings, reporting a 19 % decrease of development speed [40]. AI tools also showed positive impacts on the efficiency and self-efficacy, which is defined differently depending on the study [3, 5, 39, 46]. Oftentimes it aligned with speed, sometimes in addition to code quality or other metrics. Results on this metric also vary with studies sometimes finding improvements for some, and deterioration for other participants [48, 67].

Other notable impacts are code quality improvements [29, 39, 51, 70] and higher satisfaction [13, 39, 47, 51, 65]. However, studies discussing the drawbacks of AI tools have also often found that lacking code quality [5, 13, 29], code accuracy [5, 65], overall wrong answers [8], and resulting lack of trust [8, 33, 46, 47, 51, 65] are some of the most often mentioned issues developers have with the tools. Another relevant finding, though not frequently highlighted, is reported in one of the most comprehensive studies to date (the DORA Report 2024 [51]), which observed that increasing AI adoption is associated with higher code instability.

While these are some of the more commonly impacted areas, they vary across studies, many of which also highlight additional effects. Overall, studies paint a mixed picture: AI tools oftentimes greatly support developer's work at best, and do nothing at worst, with no generalized trend in view, and negative impacts rarely discussed. While that could indicate that AI tools are just very good at what they do, it also raises the question if negative effects are favorably glossed over. Many studies that use substantial real-world usage data are funded by corporations and therefore reflect, at least to some degree, vested financial interests (i.e., selling a product). This concern is further supported by GitClear [24], who, to our knowledge, analyzed the largest volume of code lines regarding the impact of AI. They found that although more code is being

generated and added to projects, there is also an increase in copy-pasted code, code churn, defect rates that lessen code longevity, and a decline in code reuse and refactoring. These issues are rarely investigated in other studies, likely because few analyze codebases at this scale. Additionally, reported perceptions of AI-assisted development vary, sometimes even within the same study. Together, these factors underscore the inconclusive nature of current productivity research. In this study, we aim to contribute a clearer picture by empirically analyzing real-world objective data from developers' day-to-day workflows.

2.2.2 Repository-based productivity

Beyond measuring the impacts of coding-assistants on developer's produced code and their behavior with repositories, there has been a lot of research on how we can measure productivity using repositories in general. In this research, we usually see different combinations of metrics being verified against measurements that are not code based to ensure their correctness. In this subsection, we will provide a brief overview of these works.

A common insight across studies is that individual metrics are often insufficient on their own; instead, combining multiple indicators and interpreting them within context is essential for meaningful assessments [35, 44].

Developer productivity. Several studies have attempted to quantify individual developer productivity using data from version control systems. They typically rely on code-level metrics (e.g., number of commits, number of changed lines of code, referred to as LoC) and often combine them with qualitative assessments to validate their reliability.

Oliveira et al. [44] evaluated code-based (e.g., source LoC) and commit-based (e.g., number of commits, committed LoC) metrics per developer over time. Comparing these to team leaders' perceptions, they found commit-based metrics correlated poorly, while code-based metrics aligned more closely with perceived productivity. Similarly, Lima et al. [35] validated their contribution-based metrics using team leader feedback. They considered added, deleted, and modified LoC, code complexity, and bug-fix frequency. While contribution and complexity metrics were well-received, bug-related metrics drew criticism.

Amit and Feitelson [2] propose the Corrective Commit Probability (CCP), the proportion of corrective commits, as a code quality metric, finding that a lower CCP correlated with higher developer productivity, measured by commits per developer.

When shifting from individuals to teams, studies show that even factors such as team size (number of collaborators in a repository) can influence productivity in varying ways. Some find positive effects with larger teams [28, 34],

others report negative impacts [55], and some suggest that medium-sized teams, particularly under nine members, may strike a balance [52].

Overall, research on individual metrics shows no consensus on a single best measure for productivity, underscoring the need to combine multiple metrics and contextual interpretations. These findings also underscore the importance of analyzing productivity across different levels (individuals, teams, and repositories), as results may vary and influence one another depending on the use case.

Repository Productivity. Beyond individual productivity, several studies have assessed productivity at the repository level, which is the focus of the second part of this thesis.

Liao et al. [34] developed a software ecosystem productivity model to explain factors influencing productivity, distinguishing between primary productivity (e.g., commits, pull requests) and secondary productivity (e.g., merges, comments). With their model, they identified the contributor number and activity as key factors for productivity.

Maddila et al. [37] created a tool with the goal of accelerating pull request completion as they found that overdue pull requests significantly impact communication and hinder development speed by causing negative side effects, such as extensive merge conflicts. While this does not focus on productivity, the affected measures are commonly associated with productivity (e.g., develpment speed [2, 6]), thus improving the time, until pull requests are closed, impacts productivity.

Despite ongoing criticism, LoC, or metrics based on LoC, have been a longstanding and frequently used proxy for productivity, both on individual and repository levels, with applications in code-level metrics dating back to 1981 [32].

We derived our set of 14 metrics from the research presented in this section, and share the metrics we used in more detail in subsection 3.2.2.

Chapter 3

Methodology

This chapter outlines our research methodology. We begin by presenting our research questions and their operationalization in section 3.1. In section 3.2, we describe the repository metrics and detail their computation. Then, we describe how we approached the questionnaire analysis by explaining our grouping process in section 3.3. We continue with section 3.4, where we detail the hypotheses that guide our data interpretation. Next, in the study setup (section 3.5) we give a general overview of the companies and participants we worked with as well as our study procedure. We conclude the chapter in section 3.6 with a description of our materials, including the questionnaire, data sources from both industry and open-source repositories, the data structure, and data collection tools.

3.1 Research Questions and Operationalization

In this thesis, we analyze the influence of LLM-based coding assistants on productivity from two perspectives: individual developers and entire repositories. We aim to explore whether AI-assisted development (i.e., software developers including these AI tools in their development workflows) influences repository-based metrics, revealing short-term effects on developers and long-term trends in the codebase.

To capture both short-term individual- and long-term repository-level effects on productivity, we frame one overarching research question and three subquestions:

RQ$_1$ How do LLM-based coding assistants influence productivity in real-world software projects?

- **RQ$_{1.1}$** How does AI-assisted development affect short-term developer productivity in industry repositories?

RQ$_{1.2}$ How does AI-assisted development affect long-term repository productivity in open-source projects?

RQ$_{1.3}$ How do the productivity effects of AI-assisted development compare between short-term industry and long-term open-source contexts?

RQ$_{1.1}$ and **RQ$_{1.2}$** investigate the overall impact of introducing LLM-based coding assistants on developer productivity in IRs and repository productivity in OSRs. Viewing individual productivity in IRs offers a fine-grained, developer-focused perspective over shorter timeframes, reflecting immediate productivity shifts. In contrast, the data we retrieved from OSRs, which are commonly used in empirical software engineering research, provides a broader, longer-term perspective. It smooths individual variability and captures sustained or evolving trends following AI tool adoption, as fluctuations in individual activity are averaged out across contributors as well as over time.

Answering these research questions provides tangible real-world insights into the impact of AI tools using empirical data. As such, we move beyond short-lived experiments and perceived effects. Employing this dual perspective allows us to bridge the gap between the individual developer and the broader repository productivity, reflecting the varied ways productivity manifests across contexts. Additionally, the two perspectives on time allow us to discover both immediate and long-term impacts of AI usage. It helps clarify whether claims of increased productivity from AI tools – often based on self-perceptions or controlled experiments – are supported by repository and developer data. It also clarifies which dimensions of productivity are most affected, and which trends go beyond situational effects.

To answer **RQ$_{1.1}$** and **RQ$_{1.2}$**, we compute 14 abstract, privacy-preserving metrics based on the data of 14 developers working across 39 industry repositories from a private company, as well as a set of 6 open-source repositories. For each metric, we measure changes before and after the introduction of an AI tool, using either developer-specific access dates (IR) or the release date of a commonly used LLM-based assistant (OSR). We calculate standardized effect sizes, averages and slopes to enable normalized comparisons across heterogeneous repositories and metrics within either IRs or OSRs.

In industry settings for **RQ$_{1.1}$**, we tracked developer productivity changes over shorter timeframes, aligned to individual tool adoption dates, to capture immediate behavioral shifts. We also collected demographic data and participants' expectations prior to the tool usage within an initial questionnaire. Based on effect size patterns across metrics, we grouped participants into four groups and aggregated their questionnaire responses to explore potential links between individual effects and perceptions of the tool.

In open-source settings for $\mathbf{RQ_{1.2}}$, we analyze repository productivity over longer observation windows to discover long-term trends. As we do not analyze individual developers in this setting, no questionnaire data was collected.

To guide our confirmatory analysis, we derived five hypotheses from related work, which we test using the observed metric changes for both subquestions. In addition, our exploratory analysis of the calculated metrics allows us to formulate new hypotheses about how AI tools may influence productivity dimensions in ways we had not yet thought of.

$\mathbf{RQ_{1.3}}$ investigates whether the effects of LLM-based coding assistants differ between short-term developer productivity in IRs and long-term repository productivity in OSRs. Specifically, it investigates whether individual short-term productivity changes are sustained over time, evolve into new, overarching patterns, or differ entirely. Beyond characterizing temporal and contextual variation, this question addresses the generalizability of observed effects. It does so by exploring if productivity gains in one environment translate to another. Since open-source and industry development can differ in structure and collaboration patterns, a direct comparison reveals whether LLM-related effects are robust or context-dependent.

To operationalize $\mathbf{RQ_{1.3}}$, we compare effect sizes and aggregated trends across the 14 productivity metrics used in $\mathbf{RQ_{1.1}}$ and $\mathbf{RQ_{1.2}}$ between individual industry developers and open-source repositories. This allows us to examine how observed effects vary not only across organizational contexts, but also across different temporal scopes. Specifically, we analyze whether short-term trends observed at the developer-level (in IRs) are reflected in the long-term, more general OSR trends, or whether patterns change notably. Thus, the analysis contributes to a more nuanced understanding of how AI tools shape software development across environments. We further examine whether our hypotheses, derived from the initial two subquestions, hold across the combined contexts or only within their respective settings, providing further insight into the scope and stability of LLM-related productivity effects.

3.2 Metrics

This section outlines our approach to processing and analyzing the collected repository data to calculate 14 metrics to quantify productivity. We use the metrics to answer $\mathbf{RQ_{1.1}}$ using the IR data, $\mathbf{RQ_{1.2}}$ using the OSR data, and compare both results further for $\mathbf{RQ_{1.3}}$. In the following, we first outline our general approach to computing the 14 repository metrics to our data. Once the approach is clear, we can define and justify our productivity metrics with this context. To do so, we divide the explanations into four categories based on

the underlying data: commit-based, file-based, pull request-based, and other metrics.

3.2.1 Metric Computation Procedure

We applied a consistent procedure to compute each of the 14 metrics, which we illustrate in Figure 1. To explain our process in a general manner, we refer to each repository object (e.g., commit, pull request, or branch) as an *item*.

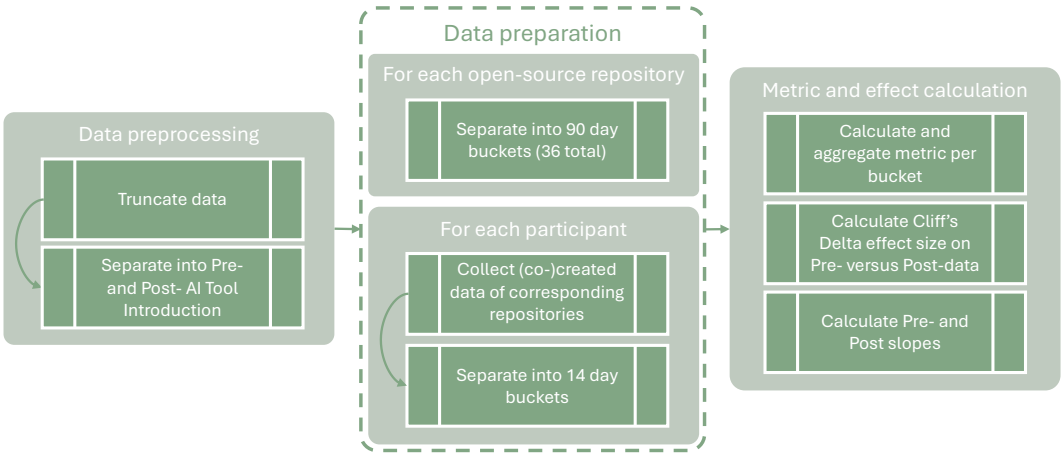


Figure 1: Process of preparing and evaluating repository data per metric

Data preprocessing. We began by preprocessing the data using the introduction date of the AI tool. For open-source repositories (OSRs), we used June 21, 2022, which is the date GitHub Copilot was released publicly following its technical preview [17]. For industry repositories (IRs), we used the individual access date reported by each participant. We then truncated the data to include an equal number of days before and after the introduction date (990 days for OSRs, 126 days for IRs) and split it into pre- and post-AI periods based on the creation timestamp of each item.

Data preparation. Our data preparation differed depending on whether the data came from an IR or an OSR.

For $\mathbf{RQ_{1.1}}$ (IR data), the analysis was conducted per participant rather than per repository. We merged all repository data and split the data by participant, resulting in one dataset per person. Each dataset included items the participant created, co-created, or was involved in. An item is included in a participant's dataset if they created it (e.g., they created a commit), were assigned to or reviewed it (e.g., they were the reviewer of a pull request), or

were mentioned in its message. For example, a commit with the message "Fix: {{Participant Name}}'s comments", would be attributed to the mentioned participant. While using item messages could lead to false positives, it helps us capture collaborative situations where multiple developers worked together, e.g., in a pair programming situation.

Each participant's data was first grouped into equal 14-day periods to capture consistent time spans across the study. We refer to these periods as buckets. We chose 14 days as the bucket size, because it falls midway within the Scrum Guide's [56] recommended sprint range (up to a month). As such, it serves as a reasonable proxy for iteration cycles while also being long enough to absorb short holidays, sick leave, and similar interruptions. The division resulted in a total of 18 buckets, with 9 of them before and 9 after the introduction of the AI-tool. All buckets follow an aligned timeline, meaning we treat them as the first, second, third, and so on bucket before or after the introduction date, rather than using specific calendar dates. This allows a consistent comparison across participants with different introduction dates.

Since we did not have access to the entire repositories' data (i.e., data from non-participants), calculating the metrics per repository would not accurately reflect repository productivity impacts. Instead, we focused on individual level productivity over a shorter timeframe, thereby offering a finer grained view of trends with AI tools to answer $\mathbf{RQ_{1.1}}$ compared to the broader OSR setup in $\mathbf{RQ_{1.2}}$.

For OSRs in $\mathbf{RQ_{1.2}}$, we separated the dataset of each repository into 90-day buckets, resulting in 11 buckets before and 11 after the AI tool's introduction. We chose a quarter year, as we judge the timeframe short enough to detect smaller changes in behaviors and productivity impact, based on discussions with our industry partners. At the same time, it is long enough to be meaningful in the broader context of long-term repository activity. It smooths over individual variation, absences, and capturing metric changes even when not all developers use AI tools. Since we could not determine who used the tool, when they began using it, or whether they used it at all, analyzing the data at the level of individual participants or in shorter timeframes was not feasible. Instead, we adopted a repository-level perspective over longer periods to reduce individual variability and better capture overarching trends potentially linked to the tool's introduction to address the question. For $\mathbf{RQ_{1.3}}$, this perspective additionally allowed us to assess whether trends observed on a smaller scale (IR trends) were short-lived, sustained over time, or evolved into new patterns altogether.

Metric and effect calculation. We calculated the metrics for each item and aggregated them by bucket using metric-specific methods, which we detail

below. We did not use the same aggregation method for all metrics to preserve the semantics and original intent of each metric.

As a result of our data preparation, we obtained 18 data points per participant for $\mathbf{RQ_{1.1}}$ and 22 for each OSR in $\mathbf{RQ_{1.2}}$, with one aggregated value per bucket per metric. We compare these pre- and post-values by calculating the Cliff's Delta effect measure. Using the effect size allows for a consistent comparison and interpretation across repositories, participants and metrics.

Cliff's Delta is a non-parametric, directional effect size, normalized between -1 and 1, and does not assume normal distribution [10]. It quantifies the amount to which two groups differ using the probability that a randomly selected value from one group will be greater than a randomly selected value from the other. As a rule of thumb, $0.147 \leq |\delta| < 0.33$, $0.33 \leq |\delta| < 0.474$, and $0.474 \leq |\delta|$ indicate small, medium, and large effects, respectively [53]. However, effect sizes should be interpreted in the context of the specific study and not solely based on such standardized benchmarks. Regarding interpretation, a positive effect reflects an increase in the metric after the introduction of the AI tool, while a negative effect reflects a decrease of the metric. However, which effect direction is desirable depends on the metric. This means that a negative effect size does not automatically imply an undesirable outcome, nor does a positive one necessarily imply a desirable effect.

Additionally, we computed total and absolute average effect sizes per metric, per person (for $\mathbf{RQ_{1.1}}$) and per OSR (for $\mathbf{RQ_{1.2}}$). We also aggregated the number of positive, negative (with $|\delta| \geq 0.147$), and missing effects for each person and OSR.

Trend Analysis. We analyze the slopes of each productivity metric before and after the introduction of AI. This approach ensures the soundness of our interpretations, provides a more nuanced view, and reveals whether AI tool adoption merely amplifies existing productivity trends or induces more fundamental changes.

To compute slope values, we apply linear regression to the bucket values of each metric before and after tool adoption. Specifically, we fit a first-degree polynomial to the available (non-missing) data points using NumPy's *polyfit*[1], where the slope coefficient represents the rate of change over time. For each metric, we compare the direction (positive or negative) and magnitude of pre- and post-introduction slopes.

We categorize each case based on whether the slope's direction is maintained or reversed between the pre- and post-introduction periods. Additionally, we assess whether the slope's magnitude remains *similar* (within 20 %

[1]https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html

of the pre-slope), becomes *stronger* (post-slope is more than 20 % higher), or becomes *weaker* (post-slope is more than 20 % lower).

This analysis reveals whether AI tools reinforce existing productivity trends or fundamentally alter development trajectories. Reversals in direction highlight cases where AI introduction corresponds with a shift in productivity dynamics, while a maintained direction shows that existing development peculiarities might be amplified or softened, ignoring the direction.

Finally, to operationalize $\mathbf{RQ_{1.1}}$ and $\mathbf{RQ_{1.2}}$, we compare the effect sizes and slopes per metric across all IRs and separately across all OSRs. To address $\mathbf{RQ_{1.3}}$, we then compare the aggregated results between OSRs and IRs.

3.2.2 Repository Productivity Metrics

We selected repository productivity metrics using a data- and constraint-driven approach. We first outlined all data that we could collect from repositories without actually directly accessing the source code and shared this with the participating companies. This allowed them to get started on discussions and NDA (Non-Disclosure Agreements) preparations with their Data Protection Department.

Next, we reviewed prior work on repository-based productivity measurements (subsection 2.2.2) and compiled a set of relevant metrics. However, after collecting data, we found that not all planned metrics could be computed equally in all companies with our collected data. For example, access to issue data (e.g., task descriptions or bug reports managed in tickets, that are usually included in version management systems) was not permitted in two of the three companies. Consequently, we refined the metric set to ensure consistency across all data sources, resulting in 14 metrics. We provide an overview of the metrics in Table 1.

Some metrics from section 2.2 could not be included in the table and the subsequent analysis. This is due to two main constraints: (i) we lacked full repository access and relied only on Git CLI and API calls, which made certain metrics, such as code duplication, unavailable; and (ii) company-specific version control systems and NDAs limited data access in varying ways.

The aggregations we apply may differ from some prior works. Each metric is explained in the following subsubsections, including its purpose and the rationale behind its aggregation.

It is important to note that abstract productivity metrics should not be interpreted in isolation. Prior research emphasizes the need to consider them collectively to form a meaningful view of productivity [35]. Additionally, some

[3] Abbreviations (Abbr.): # = number, rel. = relative, PR = pull request, LoC = lines of code, Agg. = Aggregation.

Table 1: Repository productivity metrics, grouped by the data granularity at which they are collected (e.g., commit-level metrics are calculated per commit). The table lists each metric's source, calculation, abbreviation (used in figures), and aggregation method. Empty calculations are total counts of items.[3]

Base	Metric	Calculation	Abbr.	Agg.
Commit	Coupling [2]	# of files	Coup	Mean
	LoC added [32, 35, 44, 57]	LoCa	LoCa	Sum
	LoC deleted [35, 44, 57]	LoCd	LoCd	Sum
	LoC changed [35, 44]	LoCa + LoCd	LoCc	Sum
	# of commits [2, 21, 44]		$\#C$	Count
File	Rel. code churn M1 [41]	$\frac{\text{LoCm+LoCa}}{\text{length}}$	M1	Mean
	Rel. code churn M2 [41]	$\frac{\text{LoCd}}{\text{length}}$	M2	Mean
	Rel. code churn M7 [41]	$\frac{\text{LoCm+LoCa}}{\text{LoCd}}$	M7	Mean
PR	# of successful PRs [34, 37]		#SPR	Count
	Time to close PRs [37]	$\text{Hours}_{\text{till PR closed}}$	T_{PRc}	Mean
	Time to merge PRs [37]	$\text{Hours}_{\text{till PR merged}}$	T_{PRm}	Mean
	# of PRs [21, 34]		#PR	Count
Other	# of releases [20]		#R	Count
	Time to merge to main [20]	$\text{Hours}_{\text{till merged}}$	T_{Bm}	Mean

of these metrics were originally intended for individual productivity, meaning to be calculated per person instead of per repository, and vice versa. For this thesis, we adapt them to both, individual and repository productivity, depending on whether we work with industry repositories or open-source repositories.

Commit-based Metrics

Commit-based productivity metrics are calculated on the data retrieved per commit, e.g., all LoC deleted or added within one commit. Specifically for the LoC, note that Git does not distinguish partially modified lines; such lines are typically counted as both deleted and added. The aggregation of these metrics is over all commits of a given timeframe (specified by the buckets). In the following we describe the five metrics we use:

Coupling. Coupling describes the number of files changed per commit [1], meaned over all commits in a certain timeframe.

Amit and Feitelson [1] introduced this metric as a proxy for how many files are typically involved in completing a task, based on the notion that a commit reflects such a task. They found that reductions in coupling were associated with fewer subsequent bug-fixing commits, suggesting improved code stability, which in turn positively impacts productivity.

As such we desire a negative directed effect for Coup, pointing to increased task focus.

Lines of code added, deleted, and changed. We compute three variants of LoC-based metrics: added (LoCa), deleted (LoCd), and combined (LoCc), also used to capture total code churn [41]. We calculate them (unlike in some related work) as a sum without normalization, since we only compare the actual values of the metrics within repositories and thus do not need to rely on normalization. These serve as proxies for coding activity and change volume [35, 44]. While LoC can include uncommitted code [44], we rely on committed changes due to data limitations. LoC metrics are widely used in both research and industry for their availability and perceived link to productivity [44, 57]. However, they are often criticized for their ambiguity, as more or fewer LoC do not imply better or worse outcomes in isolation [35, 49]. When interpreted in context, for example, they could help indicate broader trends such as rising or declining change rates or activity changes.

To indicate a productivity increase, it is desirable for LoC metrics to rise, i.e. show a positive effect direction, to indicate higher activity when using the AI tool. However, the ratio between the metrics also matters, as a one-sided change in added or deleted LoC may indicate unstable development, limited code reuse, or insufficient code refinement.

Total number of commits. The number of commits is formed by counting all commits in a given period. It is used to approximate development speed, which can correlate with productivity [2, 44] and one possible indicator of development activity according to the SPACE framework [21].

Similar to LoC based metrics, we expect a productivity increase, i.e. faster development speed, to be shown by a positive effect direction.

File-based Metrics

File-based metrics are derived from per-file information in each commit. To compute them, we use modified lines of code (LoCm) instead of LoCc, as the latter does not distinguish between fully added/removed lines and in-line

modifications. LoCm is generated by parsing *git diff –word-diff*[4] output and identifying added, deleted, or modified lines using pattern matching. The patterns allow us to differentiate whether a line has been added or deleted in its entirety or if only some parts of it have been added or removed (which would be considered a modified line).

However, we find that this method sometimes shows discrepancies between the resulting counts of modified, added and deleted lines and the automatically retrieved added and deleted lines per file. As such, we view the resulting file-based metrics rather as an estimation within the bounds of our capabilities as opposed to taking them as absolute truths. We take this trade-off to calculate the relative code churn rates according to Microsoft's specifications [41].

Relative code churn rates. We adapt three relative churn metrics from the metrics defined by Nagappan and Ball [41], using the same identifier as the original. M1 denotes the ratio of lines changed to total lines in the file, M2 denotes the ratio of lines deleted to total lines in the file, and M7 denotes the ratio of lines added and modified in a file to the sum of lines deleted. All three measures are calculated per file per commit and then meaned over the entire timeframe.

We combine these specific three measures because according to Nagappan and Ball [41], they can cross-check each other to allow a more nuanced view on metrics based on LoC, e.g., detecting when high churn (M1) results from feature additions (high M7) rather than deletions (M2). Microsoft originally introduced these metrics to predict defect density, as higher relative code churn was empirically linked to a greater likelihood of defects in software components [41]. Since defects can reduce productivity [64] and code churn rates in general are oftentimes raised for productivity research [24, 57], we apply these metrics as indicators of repository productivity. Consequently, we consider a decrease in code churn rates across these metrics as the desirable outcome, signaling a positive effect on productivity.

Pull Request-based Metrics

Pull requests (PRs) propose merging a source branch into a target branch. A PR can be closed without changing the target branch, or merged (successful), updating the target branch with the source branch's contents. Depending on the version control system, the latter is also considered closed. If neither state has been reached, a PR is considered open. PR-based metrics are computed per PR and aggregated over all PRs within a bucket.

[4] *git-diff* shows changes between commits, *word-diff* is used for granularity, see https://git-scm.com/docs/git-diff

Successful and total pull requests. We count all PRs, regardless of state, within a given period to obtain the total number of PRs (#PR). We also compute the number of successful (merged) PRs (#SPR). The total number of PRs is an indicator for developer activity [21, 55]. Since PRs are typically reviewed and tested by peers, they represent time investment and may slow individual work [37]. A successful PR is expected to contribute correct, high-quality code, which supports productivity [64], underscoring the value of including both metrics.

As a rise of developer activity is desirable, we expect positive effect directions to indicate a productivity increase. Preferably, we see the #SPR having stronger effects than the #PR, to indicate a higher success ratio.

Time to merge or close pull requests. For each PR, we measure the time from creation until it is merged (T_{PRm}) or closed (T_{PRc}). In other words, T_{PRm} measures the time to merge (merged PRs only), while T_{PRc} includes all closed PRs, whether merged or not.

Time is computed in hours and averaged over the bucket. Tracking both outcomes captures the efficiency dimension, because it adds a temporal dimension: PRs can use a lot of developers' time, blocking reviewers from doing something else and developers from continuing their task [37]. Thus, we measure both as an indicator of productivity, noting that a negative effect direction indicates increased productivity – reflecting less time spent on PRs.

Other Metrics

Beyond commit-, file- and PR-based metrics, we track additional indicators that extend over these boundaries.

Total releases. Git tags are references to specific commits in a repository, commonly used to mark release points. We use these tags to track release activity over time (#R). This metric captures how often releases are created by counting them within a given timeframe. The metric is inspired by DORA's deployment frequency metric [20], which we attempt to approximate using the release frequency.

Similar to other activity metrics, we want an increase in activity and as such view a positive effect direction as desirable.

While useful at the repository level, this metric is less suitable for assessing individual productivity. Releases typically result from team efforts and are often created by different developers, making it difficult to attribute them to a single individual.

Time to merge into main. We calculate the average time it takes for a branch to be merged into a target branch in hours. To achieve this, we iterate over all commits of all branches of a repository, always identifying the first commit overall or after a PR and the first subsequent PR of it. We then calculate the difference between the creation of a commit and the corresponding following PR. Finally, we take the mean of all such time differences over a bucket.

Inspired by the DORA metric *Lead Time for Changes* [20], this measures the delay between a commit (or a collection of commits) being merged into a feature branch and subsequently being merged into the main branch. In other words, it measures how long it takes for code to be considered sufficient enough for the next higher stage. Therefore a desirable productivity impact is shown in a negative effect direction, indicating that changes are merged faster.

Note that due to how git repositories are structured and which practices are in place, this metric might not always result in the intended values. Branches are often deleted after merging, and even if retained, it is not always possible to trace the original commits back to their branches, which makes it impossible to correctly measure this time.

For $\mathbf{RQ_{1.1}}$, we attribute a branch to an individual developer based on whether they were the creator, last editor, or merger. As a result, this metric does not reflect individual developer productivity as required for $\mathbf{RQ_{1.1}}$, because the structure does not support reliable attribution of branches to specific individuals. With our attribution system, contributors often fall through, as branches usually involve more people than just the creator, last editor, and merger. Moreover, the time until a branch is merged does not necessarily reflect when an individual's contribution was merged.

3.3 Participant Grouping

To better understand some of the immediate effects observed in the metrics and to give a more nuanced answer to our RQs, we used questionnaire responses to explore potential explanations for individual differences.

Specifically, we analyzed participants' self-reported expectations and backgrounds from the questionnaires to understand the varying impacts observed after the introduction of AI tools. Once we had our effect sizes calculated, we identified recurring patterns across participants and used these to group them into three categories, shown in Table 2.

Table 2: Criteria for assigning participants to groups based on predominant effect size and direction.

Group	Criterion
Positive	$\delta \geq .147$
Negative	$\delta \leq -.147$
Mixed	$\delta \geq .147$ or $\delta \leq -.147$

Participants that did not fit the criteria were considered trendless or without a group.

For this grouping step, we excluded the #R and T_{Bm} metrics due to reliability concerns that were explained in subsection 3.2.2, resulting in a set of 12 metrics. We defined four groups: participants with predominantly positive effect sizes (positive group), predominantly negative (negative group), predominantly mixed (mixed group), and those with no dominant pattern. A participant was assigned to a group if at least 75 % (i.e., 8 out of 12) of their metrics showed the corresponding effect direction with at least a small effect size ($|\delta| \geq 0.147$). As such, being without a group does not mean that these participants experienced no productivity effects at all, but that, if they experienced them, they did so in less than 8 of the metrics.

We then segmented the questionnaire responses by group and compared the median answers to identify contrasts. Due to the small sample sizes of the groups, we did not conduct statistical testing. Instead, we use these observations to support or explain our interpretation of individual differences to formulate hypotheses which, in turn, should be tested in future, focused studies.

We also attempted to apply this grouping approach to open source repositories, though the resulting patterns were less distinct; we return to this in the results section.

3.4 Hypotheses

Now that we have established our research questions and metrics, we can define our hypotheses. As an entry point into our data evaluation for $\mathbf{RQ_{1.1}}$ and $\mathbf{RQ_{1.2}}$, we formulated five hypotheses based on related work and our collected metrics. In this section, we explain each hypothesis, provide our rationale, and specify the metrics used to test it. Note that when we refer to code output in our hypotheses, we specifically mean *committed* code, not all code written by a developer.

We test each hypothesis separately with $\mathbf{RQ_{1.1}}$ and $\mathbf{RQ_{1.2}}$ results, using the metrics from before the introduction and after, as summarized in Table 3. In $\mathbf{RQ_{1.3}}$, we compare the stability of the hypotheses across time and contexts, assess their robustness and generalizability, and examine how they may need to be adapted across settings.

A number of related studies report positive effects of AI tools on development speed and activity [5, 13, 29, 31, 46, 47, 59, 61, 65]. Based on these findings, we state $\mathbf{H_1}$, which addresses the impact of AI tool usage on such measurements. We expect an increase in overall activity, as reflected in met-

Table 3: Hypotheses, which metrics are used to test them, and on which page these metrics are explained.

	Hypothesis	Metric	
$\mathbf{H_1}$	The use of an AI tool causes higher developer activity and development speed.	$\#C$	(p. 19)
		#PR	(p. 21)
		#R	(p. 21)
$\mathbf{H_2}$	The use of an AI tool leads to higher code output with a greater task focus.	LoCc	(p. 19)
		Coup	(p. 18)
$\mathbf{H_3}$	The use of an AI tool negatively impacts pull request handling time and reduces pull request success rates.	T_{PRm}	(p. 21)
		T_{PRc}	(p. 21)
		#SPR	(p. 21)
$\mathbf{H_4}$	The use of an AI tool shifts the ratio of added to removed lines of code.	M1	(p. 20)
		M2	(p. 20)
		M7	(p. 20)
$\mathbf{H_5}$	The positive and negative effects of AI tools on productivity metrics balance each other out.	All	

rics capturing total values ($\#C$, #PR, and #R).

$\mathbf{H_2}$ addresses the expectation that, while the overall amount of committed code changes increases (LoCc), the code becomes more task-focused. This hypothesis builds on our own reasoning, informed by findings on AI tools causing higher code outputs [24] and supporting developers in better solving tasks [3, 5, 12, 47, 65, 69]. To approximate task focus, we use coupling and assume that lower values indicate more focused commits [2, 16]. In summary, we expect the LoCc to increase, while Coup decreases, or at least remains stable.

We formulated $\mathbf{H_3}$ by combining several observations: reports of increased code duplication [24], developers' expectations of higher debugging effort for AI-generated code, as seen in prior studies [46, 47, 59, 65], and documented limitations of AI tools in handling complex tasks and environments [3, 13, 46]. Based on these factors, we hypothesize that such issues lead to longer resolution times and higher failure rates for pull requests, where such issues (code duplicates, bugs, overlooked complexities) are ideally noticed. We test this hypothesis using the time until a pull request was merged or closed and the total amount of successful pull requests.

$\mathbf{H_4}$ explores whether the AI tools affect the balance between added or modified and removed lines of code. Prior work has shown that using AI tools increases the code churn rate, while reducing code refactoring and reusage [24]. Developers often employ such tools for boilerplate or simple code snippets [5, 24, 46, 65], but they typically lack codebase context to produce optimal

results [3, 12, 13, 46]. We thus hypothesize that the ratio of added and modified lines relative to the deletion increases, as the AI tool generates new code that might otherwise be reused from existing sources – sources that the tool is not aware of. In other words, while we cannot directly measure duplication or reusage, we expect an increase in additions/modifications without corresponding deletions. We test this using relative code churn metrics (M1, M2, M7), which also cross-validate each other to ensure observed trends are not artifacts of confounders like feature additions, which would elevate churn regardless of AI tool usage.

Lastly, we assume that the overall effects of the AI tools are balanced between negative and positive. As discussed in section 2.2, research on productivity impacts of AI tools reports mixed outcomes. With $\mathbf{H_5}$, we hypothesize that, while individual effects may also vary, they will collectively result in a neutral effect overall. We test this by combining findings across all metrics.

3.5 Study Setup

We will now outline the companies and developers included in the study without disclosing their identities and summarize the data collection procedure.

Companies and participants. We collaborated with three medium- to large-sized companies to conduct this study. All companies operated in client-facing environments, which imposed strict privacy policies. Consequently, we were limited to analyzing abstracted and pseudonymized data. This constraint, however, reflects the conditions under which many employed developers work and is therefore appropriate for our research.

Due to delays and complications on the companies' side, this thesis analyzes data from only one of the three collaborators. In this company, we initially worked with 19 developers, most of whom had already used the LLM-based tool prior to our study. Because of missing data either before or after the tool introduction, or missing questionnaire responses, we had to remove 5 of these developers' data from our analysis. As such, we base our analysis on 14 developers.

We provide a detailed description of our participants at the beginning of our results in section 4.1, based on an initial questionnaire, where we collected demographic information and expectations of using the AI tool.

Given the small sample size, we did not conduct any statistical significance testing based on our participant data. However, all participants within the company received a standardized introduction to the tool and therefore started their usage under comparable initial conditions.

Study Procedure. Each company introduced one or more LLM-based coding assistants to their developers using its own standardized approach; this introduction was not part of our study. We then distributed an initial questionnaire in each company, with its content as detailed in section 3.6. At this point, participants already had access to the tool for different ranges of time. To ensure we could not connect any participants to any of the data, the distribution was done by one contact person at each company. Participation was voluntary, and participants could withdraw at any time. Each developer was assigned a random pseudonymous identifier (study-ID) through the contact person. Participants had 2 and 4 weeks to complete the questionnaire, depending on company preferences.

Subsequently, we visited the companies in person to collect repository data, which we pseudomized using the participants' study-IDs. We did not use data from developers that did not participate. Data analysis was conducted at our facilities.

One company intended to have the data collection done by the participants themselves. To this end, we provided them with extensive and automated scripts and explanations using a GitHub repository. Participants would then send the data to the contact person, who ensured the correctness of the data and pseudonymization. They then forwarded the data to us.

3.6 Materials

This section outlines the materials used in our study. It includes an overview of the questionnaire designed to capture developer demographics and expectations of using the AI tool, details of the industrial and open-source repositories from which we gathered data, the structure of that data, and the tools used for and within the data collection.

3.6.1 Questionnaire

We created the questionnaire in close cooperation with the industrial partners. The process is visualized in Figure 2.

When designing our questionnaire, we chose not to tailor it to each company's specific expectations in order to preserve comparability across responses. At the same time, we valued their unique practitioner insights that might not be covered in existing academic literature. As such, we chose a design process that balanced both goals: We first drafted the questionnaire based on aspects we deemed relevant to answer our RQs. We then sent it to one of our contact persons for feedback, evaluated their input, and updated the questionnaire ac-

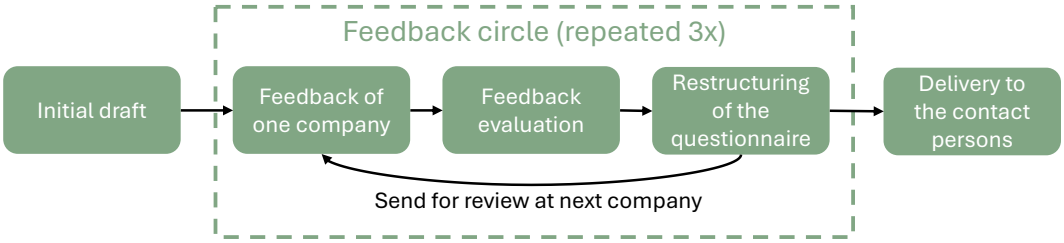


Figure 2: The linear process of creating the questionnaire in cooperation with the companies

cordingly. This process was repeated with each subsequent contact, resulting in three feedback loops, until all had provided their input. Overall, feedback decreased with each iteration, and the final round only led to minor additions to answer options for a few general questions.

In total, the questionnaire consisted of seven question-groups:

General: Categorization, e.g. study-ID and used AI tool.

Programming knowledge and experience: Programming experience without AI focus.

Experience with AI tools: General experience with AI tools focusing on private usage

Experience with AI tools at work (if applicable): General experiences with the AI tool introduced, e.g. which tool is used, how long/often it has been used, etc.

About the introduction of the AI tool: Opinion on the introduction process of the tool into the workplace, e.g. through workshops, etc.

Expectations regarding the introduction of the AI tool: Expected effects at the introduction of the AI tool.

Experience with the AI tool in everyday work: Exact mirror of the expectations but now asking for the explicit experiences.

About you: Character trait questions according to a short version of the Big Five Personality test.

Conclusion: Organisational closing questions.

The study in this thesis is part of a larger project that will utilize all of the collected survey replies. However, we require only three of the question groups to contextualize our findings regarding repository productivity. We use the *General* information to connect survey responses with repository data and check general external factors that could impact their behavior and opinions, the section *programming knowledge and experience*, to describe and understand our participants, and *expectations at the introduction of AI tools*, to assess if the expectations had an impact on the results. As such, we will describe only

these sections more detailed. In Appendix A, we provide these questionnaire sections in detail, with all questions and answer options included.

The *general* information we asked from participants included three aspects: their study-ID, the AI tool they use, and their team size (which has been shown to impact productivity [52, 55]). This information allowed us to link repository data to survey responses and consider a small selection of external factors that might influence participants' perceptions and behavior, even if they were not consciously aware of them.

The section *programming knowledge and experience* collects participants' programming background and experience. For assessing it, we used the guidelines proposed by Siegmund et al. [60] as a baseline, updating them to fit the professional environment of a software developer. We then added and removed questions and answer options based on the feedback of the contact persons. The section includes questions on education, years of programming (private and professional), self-assessed proficiency, primary programming languages, IDEs, and frameworks used.

Expectations at the introduction of AI tools included three matrix questions derived from the answer options in the 2024 Stack Overflow Developer Survey [46], covering anticipated benefits, potential challenges, and expected usage areas. The third matrix also incorporates findings from Mailach et al. [38], feedback from the companies, and frequently mentioned domains in other related work. Additional open-ended questions address expected effects with the AI tool, new tasks, and scenarios where participants would avoid using the tool.

3.6.2 Industry Repositories

We collected data from repositories of three companies, with the analyzed data being produced by or in cooperation with the participants of this study. Data created exclusively by non-participating developers was not used for the analysis. For instance, a pull request created, closed and reviewed by a non-participant, is not included in the collected data. However, if one of the roles (creator, reviewer, or closer) was covered by a participant, it would be. We provide an overview of the collected data volume in Table 4.

From the first company, we collected data from 3 repositories, amounting to a total of 9519 commits. However, due to delays on the company's side, this data is not analyzed in this thesis. Since we do not have access to the code content or the total lines of code per repository, we report commit counts to indicate data volume.

From the second, we collected data from 39 repositories, with a total of 20569 commits. This company's data forms the basis for the analysis within

Table 4: The collected industry repository data from the three companies.

	Participants	Repositories	Commits	Analyzed
Company 1	10	3	9519	no
Company 2	19	39	20569	yes
Company 3	5	ongoing	ongoing	no

this thesis. This is more than double the commits compared to the first company, which is a result of two factors: First, the second company's microservice architecture assigns one service per repository, leading to a larger number of repositories. Second, their study group includes over twice as many participants, increasing the available data.

For the third company, the data collection was not done manually by us. Instead, we provided fully automated scripts that extract and anonymize all the required data to the company's spokesperson, allowing them to collect the data on their own. We have so far received data from three repositories involving two participants. However, since the data collection is still on-going, we omit this company's data for this thesis and will review it at a later point.

3.6.3 Open-Source Repositories

For the analysis of OSRs, we selected six open-source repositories from GitHub, licensed under the MIT License, which allows free unlimited access to the software[5]. We present the repositories and some confounding data in Table 5

Table 5: Selected open-source repositories, including their abbreviations used in graphics, main programming language, domain, total LoC, and star count (according to their GitHub pages as of May 26, 2025).

Abbr.	Repository	Main Lang.	Domain	Size	Stars
VSC	VS Code	TypeScript	Code Editor/IDE	∼2.5M	154k
BS	Bootstrap	JavaScript	Web Framework/UI	∼160k	166k
RN	react-native	JavaScript	Mobile App Framework	∼2M	116k
FA	fastapi	Python	Web Framework/API	∼60k	67k
GIN	gin	Go	Web Framework/API	∼55k	76k
GD	godot	C++	Game Engine/Editor	∼1.6M	81k

These repositories were identified using GitHub's integrated Copilot Chat and were chosen based on several criteria: we selected two repositories (Visual

[5]https://mit-license.org

Studio Code and Bootstrap) due to their extreme popularity, shown by over 150 thousand stars. We selected four additional ones based on continuous activity between October 2019 and March 2025, relevance to software development, and diversity in programming languages. This selection was intended to ensure coverage across different technical stacks and to complement the company analysis with publicly available development projects. Visual Studio Code (VS Code) has the additional advantage of being the first IDE to offer full GitHub Copilot support, which we assume increases the likelihood of widespread AI tool usage among its developers.

3.6.4 Repository Data

We collected metadata from version control systems along four dimensions: branches, commits, pull requests, and releases. The following list summarizes the detailed collected data:

Branches: Branch name, number of commits, creation date and author, first and last commit SHAs (unique IDs associated with a commit), last activity date, last author, and whether the branch was merged.

Commits: SHA, author, timestamp, message, parent SHAs, lines of code added and deleted. Per-commit file-level data: file path, file SHA, raw and calculated line additions/modifications/removals, total changed lines, and file length (in lines).

Pull requests: Merge ID, associated SHA, author, merge author, timestamps of the creation / last update / closure / merge, title, description, requested reviewers, labels, assignees, and duration between creation and closure / merge.

Releases: Tag, associated SHA, author, timestamp, and message.

Not all of the included sub-items were used in our metrics or analysis due to inconsistencies across version control platforms or them not suiting any of the relevant metrics. We collected them anyways for future research.

3.6.5 Data Collection Tools

We used two tools and technologies to collect the data evaluated in this thesis.

Limesurvey. For launching the questionnaires in a pseudonym (since Study-IDs are linked to repository data) and secure manner, we used a *Limesurvey* instance hosted by Leipzig University[6]. Limesurvey is a browser-based tool for online surveys and evaluation which contains a selection of question types that

[6]https://www.urz.uni-leipzig.de/en/service-catalogue/servicedetail/service/online-evaluation-and-surveys-with-limesurvey

can be used to structure the survey, as well as a secure and DSGVO conform manner of collecting replies.

Version control systems. Version control systems are tools that track and manage changes to source code over time, enabling collaboration and preserving project history. For the repository data retrieval, we accessed repositories from four such systems: Github[7], GitLab (privately hosted)[8], GitBucket (privately hosted)[9], and Azure DevOps Repositories[10].

All OSRs were retrieved from GitHub, which was selected due to its accessibility and widespread use in research. The other platforms were mandated by whichever version control managers were used within the companies. While all of these are git-based managers, the API, structure and, in some cases, contents of the repository data differs. As such, we took care to focus our data retrieval on the underlying git-based system as much as possible and only used API-calls when absolutely necessary. The scripts and code to access the systems can be found in the supplementary material[11].

[7] https://github.com
[8] https://about.gitlab.com
[9] https://bitbucket.org/
[10] https://azure.microsoft.com/en-us/products/devops/repos
[11] https://github.com/AnnemarieWittig/supplementary-masters-thesis/

Chapter 4

Results

In this chapter we share the results of the study we conducted for this master's thesis. The underlying data and analysis scripts can be found in the supplementary material (see footnote 11).

We begin the evaluation by presenting descriptive statistics of our participants' developer experience, as collected through the initial questionnaire. Next, we address our research question by presenting our results for each of the three subquestions in separate sections, closing each with a summary of the findings. Interpretations and implications derived from these results will be discussed in the next chapter.

We will focus a lot on directional effect sizes, which can be positive (>0) or negative (<0). As we explained for each metric, we want to point out again that a negative effect size does not always imply an undesirable outcome, nor does a positive one necessarily imply a desirable effect. Instead, effects should be judged in the context of each metric.

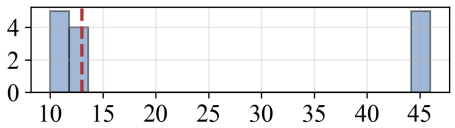
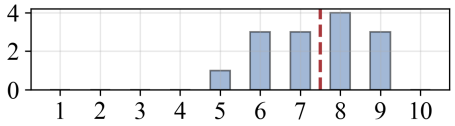
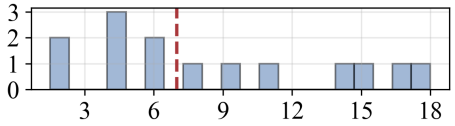
4.1 Descriptive information

This section summarizes participants' backgrounds and expectations, based on responses to the initial questionnaire.

Starting with the education, all participants had completed at least their A-levels but varied in professional experience. We give an overview of participants' replies regarding professionel experience in Table 6.

The median professional programming experience was 13 years, with a few notable outliers on both ends. However, the median time since they began working on larger software projects was notably shorter with 7 years. When asked to rate their programming experience on a scale from 1 (inexperienced) to 10 (expert), responses ranged from 3 to 9, with a median of 7.5. Participants also compared their experience to that of colleagues at their company, within

Table 6: Descriptive statistics of the participants, collected through the questionnaire responses, showing the median (M) and distributions of responses; histograms show the metric on the x-axis and the respective count of participants on the y-axis.

	M	Distribution
Years of professional programming experience	13.0	
Self-rated programming experience (on a scale of 1-10)	7.5	
Years of private programming experience	7.0	
*Programming experience... (5-item likert from **I**nexperienced to **E**xperienced)*		
...compared to all colleagues	3.0	
...compared to team colleagues	4.0	
...compared to experts	3.0	

their team, and to experts with 10 years of experience. In all cases, many perceived themselves as being at a similar level (50 % compared to all colleagues, 43 % for the other two comparisons), though a substantial portion, sometimes more than half, rated themselves as more experienced.

Regarding their general work contexts, participants reported working in teams of different sizes, with 50 % (7 participants) in medium-sized teams of 4–7 developers, 42 % in larger teams, and 7 % in smaller teams. At the time of the questionnaire, they had access to an AI tool for between 3 and 5 months. The majority used ChatGPT, GitHub Copilot or JetBrains AI assistant, or a combination of the three, typically within a development environment from JetBrains or Visual Studio Code. Aside from some exceptions, participants primarily programmed in TypeScript/JavaScript and Java, with most using the frameworks Angular and Springboot.

Overall, the developer population in our study reflects a typical company setting, with diverse experience levels but relatively consistent use of tools and programming languages.

4.2 $RQ_{1.1}$: How does AI-assisted development affect short-term developer productivity in industry repositories?

To share our results regarding the impacts of AI tools on industry repositories and their developers, we will first explore the repository metrics in general, presenting trends and patterns visible in the data. Afterwards, we contextualize these findings using participants' questionnaire responses.

4.2.1 Repository Results

We computed repository-based productivity metrics separately for each participant and visualized the results in Figure 3.

As anticipated, metrics in the *Other* category (total number of releases #R and time until a branch is merged T_{Bm}) are often zero or unavailable, as mapping them to individual developers results in missing data for most participants. Therefore, we exclude these metrics from our evaluation but include them in the overview to maintain consistency with the metrics reported for $\mathbf{RQ_{1.2}}$.

Across the remaining metrics, we observe 168 individual effect sizes (12 metrics x 14 participants):
22.02 % (37) are large, 15.48 % (26) are medium, 40.48 % (68) are small, and 22.02 % (37) show no effect. We use the terms signed and absolute to distinguish between the direction-sensitive mean (Avg) and the direction-agnostic mean (|Avg|), respectively, to avoid confusion. The high proportion of at least small effect sizes results in the average absolute effect size being at least small for every metric. However, the mean (signed) effect sizes are small at best and often close to zero, indicating balanced effects in both directions that cancel each other out on average.

The commit-based metrics show no distinct patterns beyond overall averages. Effect sizes for the number of commits ($\#C$) often align with those for codeline metrics (LoCa, LoCc, LoCd), i.e., if one is negative, the others are likely to be as well, though not consistently. Coupling differs in direction from $\#C$ in about half the cases, meaning that more commits often result in them being more focused and reduced commits therefore often lead to more files per commit.

The file-level metrics show the lowest absolute average effects, with medium effects being rare and only three large ones observed. Relative code-churns show no notable patterns, nor is there a clear relationship to other commit-based metrics, despite both being calculated from commits.

Figure 3: Metric trends per participant before and after the AI tool's introduction (IR data). Cells show effect size (bold, –1 = decrease, +1 = increase), pre-AI slope (middle), and post-AI slope (bottom). Rows = participants; columns = metrics. Final rows show average (Avg) and absolute average (|Avg|).

The pull request metrics show the highest average (signed) effects, which are always at least small, and the highest absolute average effects. This is reflected in 96 % of PR effect sizes being at least small effects, a much higher proportion than across all metrics. Overall, the total PRs (#PR) and successful ones (#SPR) tend to, most often, decrease by similar magnitudes, indicating fewer pull requests and, consequently, fewer successful ones after introducing AI tools. In contrast, the time to close (T_{PRc}) and merge PRs (T_{PRm}) generally increases, though, again, with similar strengths. This results in the interesting observation that these pairs often move in opposite directions, i.e., a decrease in the number of pull requests is frequently accompanied by an increase in the time they remain open.

For instance, participant P8 shows a strong decrease in #PR and #SPR, with both effect sizes at –0.67, and a small increase in T_{PRc} and T_{PRm} with effect sizes of 0.20 and 0.22 respectively. To illustrate this in numbers, the total number of PRs dropped from 162 to 73 and successful ones from 161 to 72. The average time to merge increased from 0.91 to 4.6 hours, and the average time to close is essentially the same, as nearly all PRs were merged. The exact values for each bucket can be seen in our supplementary material.

To identify whether the picked up effects were due to the AI tool's introduction and not just a continued trend of the behavior participants already showed before the introduction, we analyzed the slopes more in depth by comparing their direction and magnitude before (middle row of Figure 3) and after (bottom row) the tool's introduction. We provide an overview of the slope changes and their strength in Table 7.

We find that trend directions are maintained in 57.1 %, although by different margins. More exactly, with the same direction, the magnitudes are similar (i.e., in a 20 % radius from each other) for only 6 % of all measurements, weaker (i.e., over 20 % less) for 26.8 %, and stronger for 24.4 %. Additionally, we find that 41.7 % of the slopes are reversed, meaning they either switched from negative to positive or vice versa. As such, there is no overarching tendency of continuing, strengthening, weakening, nor turning already existing movements.

Table 7: Proportion of trend directions maintained or reversed before and after the AI tool's introduction in the IRs. Similar trends vary by less than 20 % in magnitude; stronger/weaker trends differ by over 20 %, regardless of direction.

Direction	Maintained	Reversed
Similar	10 (6.0 %)	6 (3.6 %)
Weaker	45 (26.8 %)	44 (26.2 %)
Stronger	41 (24.4 %)	20 (11.9 %)

An interesting case arises when both pre- and post-introduction slopes are negative, yet the overall effect size is positive, or inversely. For example, this appears for participants P2, P4, and P5. To understand this occurrence, we visualized their commit activity (#*C*) throughout the buckets in Figure 4. For

all three participants, the pre- and post-introduction trends (green) are negative, while the total trends (purple) are positive. This indicates that although the commit frequencies themselves show a decreasing tendency if viewed separately, there is a jump in commits immediately after the tool's introduction, raising the overall count to a generally higher level. The Cliff's Delta effect size measures how often post-introduction values exceed pre-introduction ones, while slope magnitudes reflect local trends rather than overall level differences. As such, a sudden jump in values can produce a positive effect size even when the slopes would suggest different.

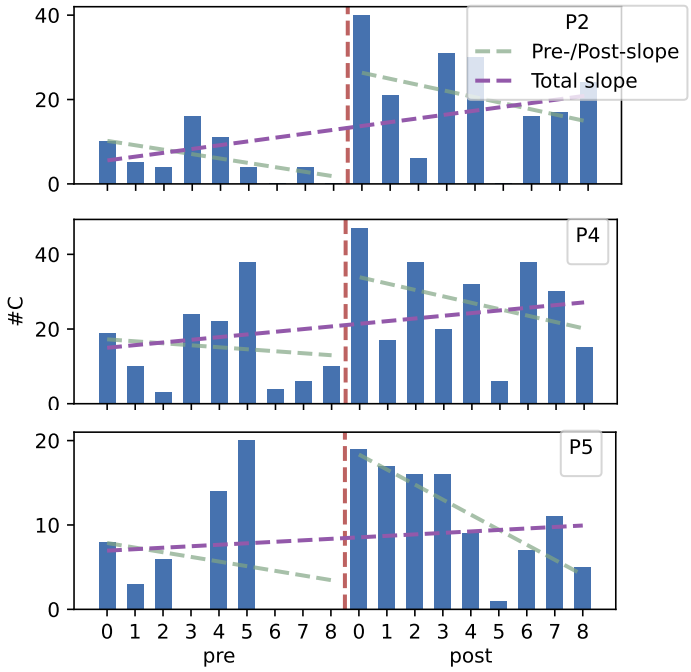


Figure 4: Participants P2, P4, and P5's $\#C$ over all buckets, including their total, pre- and post-introduction slopes.

Another notable pattern appears in metrics like Coup for P3 and P4, where slopes reverse from positive to negative, yet the overall effect size remains positive. Before the AI tool's introduction, these participants' Coup values are mostly low, though at an upward trend. After the introduction, values are generally higher but gradually decrease over time, leading to a negative slope. Despite this decline, the overall trend across the full timeframe is positive, resulting in a positive effect size.

If we turn our focus from the metric- perspective (columns) toward the participants (rows in Figure 3), some patterns seem to emerge more clearly.

Specifically, we find that around half of the participants tend to have the same effect directions for most of their metrics. Two prime examples of this are P1 and P2. Both display notable effects, but P1 in the negative direction in 10 out of 12 metrics, while P2 does so in the positive in 12 out of 12 metrics, albeit with smaller effects. To make this effect more clear, we compiled Table 8, providing a summary of the number of directional effect sizes and the average size per participant.

Table 8: Summary of participant's effect sizes (based on IR data), showing the number of positive ($\delta \geq 0.147$), negative ($\delta \leq -0.147$), and negligible ($|\delta| < 0.147$) effects; and average signed and absolute effect sizes.

	$\delta \geq 0.147$	$\delta \leq -0.147$	$\|\delta\| < 0.147$	Avg δ	Avg $\|\delta\|$
P1	2	10	0	-0.45	0.66
P2	11	0	1	0.39	0.39
P3	3	4	5	-0.02	0.24
P4	5	3	4	0.22	0.37
P5	10	0	2	0.33	0.36
P6	4	5	3	-0.07	0.23
P7	2	6	4	-0.15	0.25
P8	6	3	3	-0.06	0.29
P9	4	5	3	0.01	0.24
P10	2	9	1	-0.28	0.34
P11	3	5	4	-0.03	0.19
P12	4	5	3	0.04	0.34
P13	0	9	3	-0.16	0.20
P14	2	9	1	-0.13	0.35
Total	58	75	62	-0.02	0.32

Using these numbers, we can find four patterns: participants with predominantly positive (*positive* group), predominantly negative (*negative* group), predominantly mixed (*mixed* group) effect sizes, and those with no effects for most metrics (trendless group). After segmenting these groups according to the approach explained in section 3.3, we find there are 2 participants fitting the positive pattern, 4 fitting the negative, 7 fitting the mixed, and only one person remains without at least consistently small effect sizes. We highlighted the respective participants in the table. Interestingly, the mixed group still displays more negative effects than positive.

Regarding overall effects, we find that half of the participant show a medium to large average effect (Avg $|\delta|$), and all others a small one. The average direction of effects tends to be negative (Avg δ), indicating that negative effects tend to be bigger.

4.2.2 Questionnaire Responses

Using the participant groups, we separated the questionnaire responses regarding their affiliation to the defined patterns to compare whether any of their responses differ strongly enough to potentially warrant the difference in effect sizes. Again, we did not perform significance testing on these replies due to the group sizes.

While we mostly focus on the difference in expectations, Figure 5 shows the median replies of participants regarding their programming experience compared to other people around them. This is one of the demographic metrics, though we re-report it here due to a noticeable observation: The trendless group, albeit only one person, consistently rates themselves as more experienced (compared to peers and experts) than the other groups. In comparison, the other three groups generally see themselves as neither clearly more nor less experienced than their peers, with upward shifts in some comparisons.

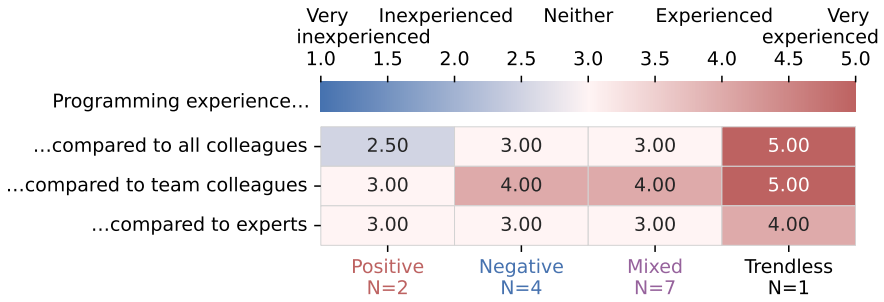


Figure 5: Median responses to self-perceived programming skills compared to colleagues at the company, in their team, and to experts with 10 years of experience.

Continuing with two matrix questions on concerns and improvements related to the AI tool, we visualized participants' median responses in Figure 6. Positive participants express the strongest concerns (first matrix), especially regarding untrustworthy answers (4.0) and security risks (4.5). Negative participants show less apprehension, with most answers disagreeing with the negative statements. In contrast, the mixed group takes a more ambivalent, yet generally positive stance. Notably, all groups agree that the tool may produce bad answers due to the lack of context about the codebase.

Most of the participants agree with the positive expectations (second matrix in Figure 6) or remain neutral at worst. Only the trendless participant stands out, indicating that the tool will not be helpful to organize their workload, while simultaneously agreeing completely that it will produce higher-quality code than themselves.

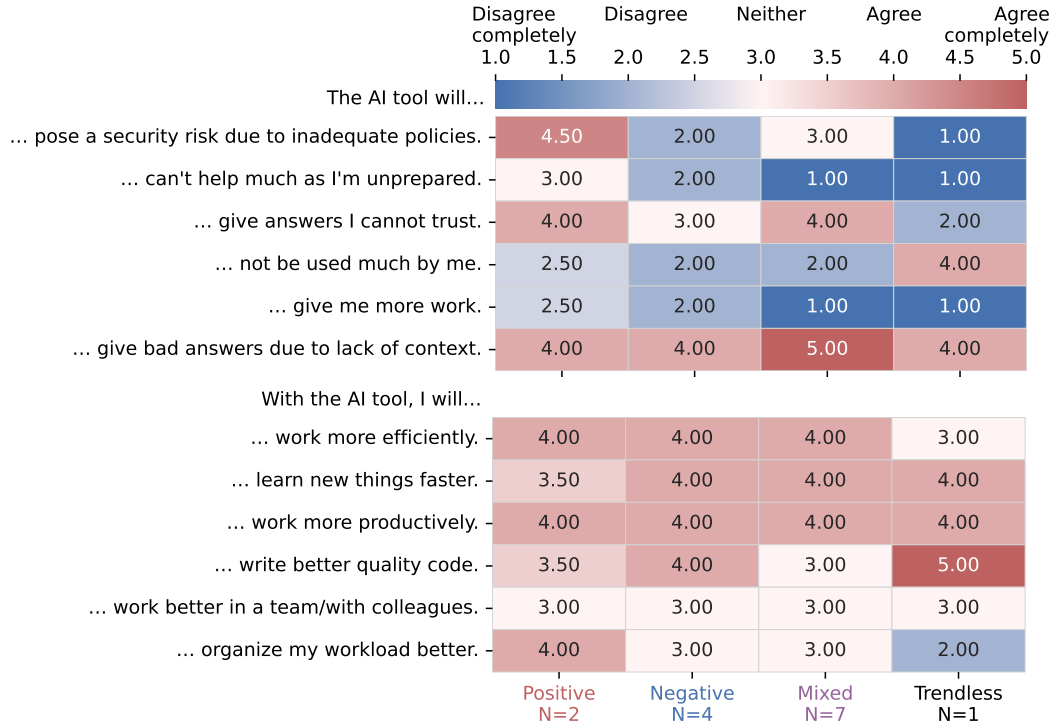


Figure 6: Median responses to concerns and perceived benefits of using the AI tool.

Figure 7 illustrates how frequently participants expect to use the AI tool for various programming-related tasks. Across most items, autocompletion and code generation rank among the most anticipated use cases, receiving high scores from all groups, while code documentation and task structuring is expected to occur the least.

The trendless participant stands out with consistently higher usage expectations across nearly all tasks, including categories, where other groups express lower or more selective interest, such as code debugging and task automation. In contrast, the positive, negative, and mixed groups show varied expectations depending on the task. Positive participants expect to use the tool more frequently overall, with three tasks rated between sometimes and often, three rated often to daily, and only one rated sometimes. In contrast, Negative participants expect lower usage, rating four tasks as sometimes and three as often. Both groups rate seven tasks as rarely or never. Mixed participants remain more neutral, typically selecting sometimes for about half the tasks and often for only one.

Overall, while less clear than participants' effect sizes, the results reveal group-specific expectations toward AI integration in programming workflows, including its benefits, risks, and usage intensity.

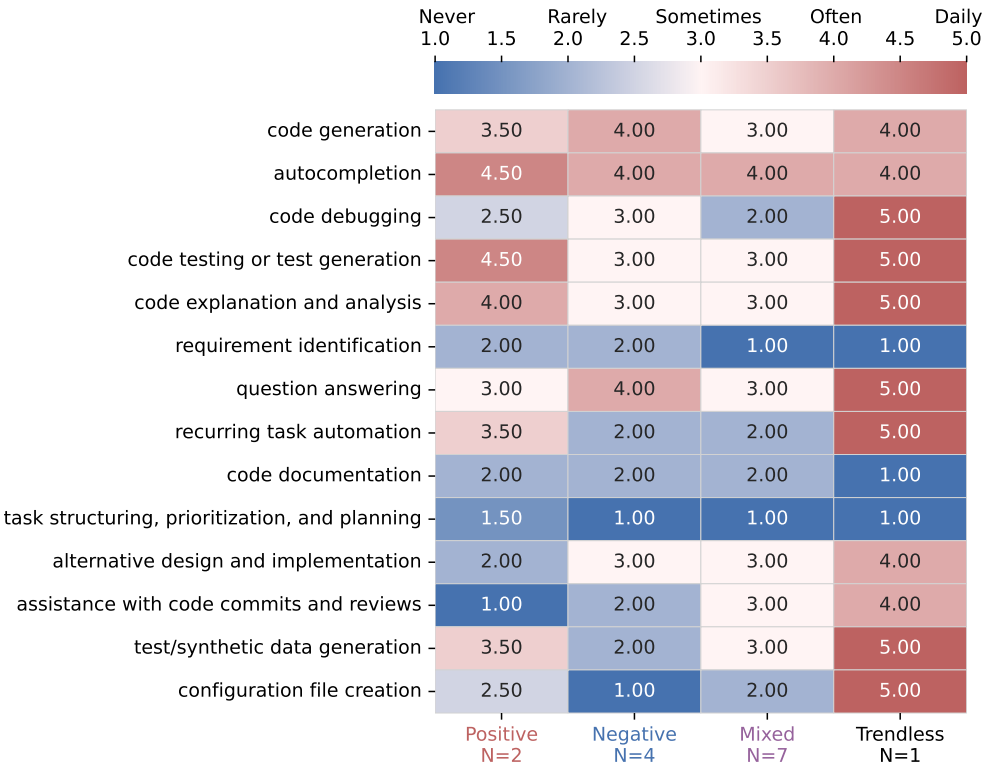


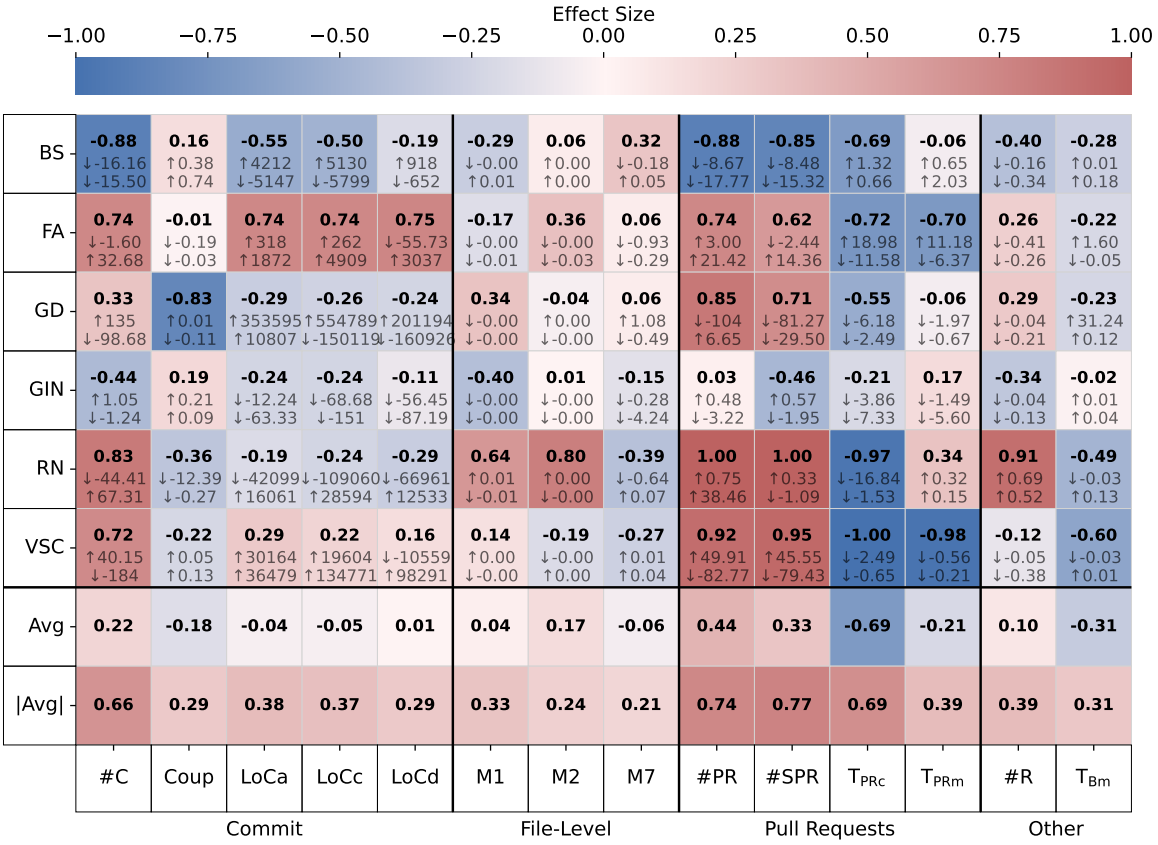
Figure 7: Median responses on expected AI tool usage per task.

Summary: $RQ_{1.1}$ findings

Most productivity metrics show small to medium effect sizes after the AI tool's introduction, with 78% of measurements indicating at least a small effect. Pull request metrics exhibit the strongest and most consistent changes, oftentimes moving in pairs in the same directions by the same magnitudes, while file-level metrics show the weakest, without any pairwise trends. Slopes are maintained in 57% of the cases and reversed in 42%, with no consistent strengthening or weakening trend. Despite varying magnitudes, some participants display consistent directional trends, forming distinct positive, negative, and mixed-effect groups.
These groupings align with patterns in questionnaire responses. Positive participants voice stronger concerns about AI reliability and security but anticipate moderate use for multiple tasks. Negative participants express fewer concerns and anticipate lower usage. Mixed participants are generally neutral but predict the least usage overall. The trendless participant stands out with unusually high self-assessed expertise and higher-than-average expectations across nearly all usage dimensions. Though groups are small, the findings suggest a potential alignment between participants' self-perceptions, usage expectations, and observed effects.

4.3 $RQ_{1.2}$: How does AI-assisted development affect long-term repository productivity in open-source projects?

We computed repository-based productivity metrics for six open-source repositories to assess the impact of LLM-based coding assistants on open-source repositories. The data and result structure follow the same format as presented in section 4.2 for **$RQ_{1.1}$**, but based on entire repositories rather than individual participants. Since we calculated the metrics over all the data of a repository, all 14 metrics were computed successfully and with sufficient data. We visualized the effect sizes per repository, along with their pre- and post-introduction trends, in Figure 8.



	#C	Coup	LoCa	LoCc	LoCd	M1	M2	M7	#PR	#SPR	T_{PRc}	T_{PRm}	#R	T_{Bm}
BS	**-0.88** ↓-16.16 ↓-15.50	**0.16** ↑0.38 ↑0.74	**-0.55** ↑4212 ↓-5147	**-0.50** ↑5130 ↓-5799	**-0.19** ↑918 ↓-652	**-0.29** ↓-0.00 ↑0.01	**0.06** ↑0.00 ↑0.00	**0.32** ↓-0.18 ↑0.05	**-0.88** ↓-8.67 ↓-17.77	**-0.85** ↓-8.48 ↓-15.32	**-0.69** ↑1.32 ↑0.66	**-0.06** ↑0.65 ↑2.03	**-0.40** ↓-0.16 ↓-0.34	**-0.28** ↑0.01 ↑0.18
FA	**0.74** ↓-1.60 ↑32.68	**-0.01** ↓-0.19 ↓-0.03	**0.74** ↑318 ↑1872	**0.74** ↑262 ↑4909	**0.75** ↓-55.73 ↑3037	**-0.17** ↓-0.00 ↓-0.01	**0.36** ↓-0.00 ↓-0.03	**0.06** ↓-0.93 ↓-0.29	**0.74** ↑3.00 ↑21.42	**0.62** ↓-2.44 ↑14.36	**-0.72** ↑18.98 ↓-11.58	**-0.70** ↑11.18 ↓-6.37	**0.26** ↓-0.41 ↓-0.26	**-0.22** ↑1.60 ↓-0.05
GD	**0.33** ↑135 ↓-98.68	**-0.83** ↑0.01 ↓-0.11	**-0.29** ↑353595 ↑10807	**-0.26** ↑554789 ↓-150119	**-0.24** ↑201194 ↓-160926	**0.34** ↓-0.00 ↓-0.00	**-0.04** ↑0.00 ↓-0.00	**0.06** ↑1.08 ↓-0.49	**0.85** ↓-104 ↑6.65	**0.71** ↓-81.27 ↓-29.50	**-0.55** ↓-6.18 ↓-2.49	**-0.06** ↓-1.97 ↓-0.67	**0.29** ↓-0.04 ↓-0.21	**-0.23** ↑31.24 ↑0.12
GIN	**-0.44** ↑1.05 ↓-1.24	**0.19** ↑0.21 ↑0.09	**-0.24** ↓-12.24 ↓-63.33	**-0.24** ↓-68.68 ↓-151	**-0.11** ↓-56.45 ↓-87.19	**-0.40** ↓-0.00 ↓-0.00	**0.01** ↓-0.00 ↓-0.00	**-0.15** ↓-0.28 ↓-4.24	**0.03** ↑0.48 ↓-3.22	**-0.46** ↑0.57 ↓-1.95	**-0.21** ↓-3.86 ↓-7.33	**0.17** ↓-1.49 ↓-5.60	**-0.34** ↓-0.04 ↓-0.13	**-0.02** ↑0.01 ↑0.04
RN	**0.83** ↓-44.41 ↑67.31	**-0.36** ↓-12.39 ↓-0.27	**-0.19** ↓-42099 ↑16061	**-0.24** ↓-109060 ↑28594	**-0.29** ↓-66961 ↑12533	**0.64** ↑0.01 ↓-0.01	**0.80** ↑0.00 ↓-0.00	**-0.39** ↓-0.64 ↑0.07	**1.00** ↑0.75 ↑38.46	**1.00** ↑0.33 ↓-1.09	**-0.97** ↓-16.84 ↓-1.53	**0.34** ↑0.32 ↑0.15	**0.91** ↑0.69 ↑0.52	**-0.49** ↓-0.03 ↑0.13
VSC	**0.72** ↑40.15 ↓-184	**-0.22** ↑0.05 ↑0.13	**0.29** ↑30164 ↑36479	**0.22** ↑19604 ↑134771	**0.16** ↓-10559 ↑98291	**0.14** ↑0.00 ↓-0.00	**-0.19** ↓-0.00 ↑0.00	**-0.27** ↑0.01 ↑0.04	**0.92** ↑49.91 ↓-82.77	**0.95** ↑45.55 ↓-79.43	**-1.00** ↓-2.49 ↓-0.65	**-0.98** ↓-0.56 ↓-0.21	**-0.12** ↓-0.05 ↓-0.38	**-0.60** ↓-0.03 ↑0.01
Avg	**0.22**	**-0.18**	**-0.04**	**-0.05**	**0.01**	**0.04**	**0.17**	**-0.06**	**0.44**	**0.33**	**-0.69**	**-0.21**	**0.10**	**-0.31**
|Avg|	**0.66**	**0.29**	**0.38**	**0.37**	**0.29**	**0.33**	**0.24**	**0.21**	**0.74**	**0.77**	**0.69**	**0.39**	**0.39**	**0.31**

Figure 8: Metric trends per repository before and after the AI tool's introduction (OSR data). Cells show effect size (bold, –1 = decrease, +1 = increase), pre-AI slope (middle), and post-AI slope (bottom). Rows = participants; columns = metrics. Final rows show average (Avg) and absolute average (|Avg|).

Across 14 metrics and 6 repositories, we observe a total of 84 effect sizes, 32 (38.10 %) of which are large, 12 (14.29 %) are medium, 27 (32.14 %) are small, and 13 (15.48 %) show no effect at all. The average (signed) effect sizes are equally positive and negative, with at least small, up to large, effects for half of the metrics. For the other half, positive and negative effects balance out, resulting in insignificant values. The absolute average is always at least small, reaching medium in 6 and large in 4 metrics.

The commit-based metrics show a range of small, medium, and large effects. The *#C* stands out with the strongest effects in this category: three strong and one medium positive effects, and one strong and one medium negative effect. Coupling decreases in three repositories, increases in two, and shows a negligible change in one. The line of code metrics, albeit with mostly small effects, usually move together in the same direction and by similar magnitudes. The exception is LoCd, often changing less than LoCa and LoCc, despite following the same direction. For example, the repository BS in the first row shows effects of the size -0.55, -0.5, and -0.19 for LoCa, LoCc, and LoCd respectively.

File-level metrics show the smallest effects after the AI tool's introduction, with a third of the effects being negligible. There are indications that the number of commits impacts the first relative code churn metric (M1) which represents the ratio of modified lines of code to file length. They often move in the same direction, in some cases by a similar margin (specifically for GD, GIN, and RN). For example, more commits tend to lead to a higher M1, indicating that files are being modified more intensely relative to their size, and vice versa. Many of the file-level metrics also show pre- and post slopes to have a magnitude in the 3rd decimal place or below, which makes sense considering they are a relative metric, and thus have values between 0 and 1.

The pull request metrics show the strongest effects. In most cases, changes in the number of pull requests (#PR) correspond to similar changes in successful pull requests (#SPR), with the exception of GIN, where only #SPR changes notably. The time to close and merge pull requests moves in the same direction in four out of six repositories, though only two show similar magnitudes. Notably, the time to close a pull request (T_{PRc}) decreases, i.e., PRs are closed quicker, in all but one case after AI tool's introduction, mostly with large effects. In contrast, the effects shown for the time to merge (T_{PRm}) are weaker and not consistently negative.

The *Other* metrics span the full range of effect sizes. An interesting observation is that the time until a branch is merged into main (T_{Bm}) decreases across all repositories.

We illustrate directional changes in pre- and post-introduction trends in Table 9. We find that about half of all slopes (55.9 %) maintain their direction, while the other half reverses. In both cases, changes are typically accompanied

by shifts in magnitude, with only 4.8 % of slopes showing similar strength before and after the AI tool's introduction.

Slope patterns are similar to those observed in **RQ$_{1.1}$**, including cases where slope directions are maintained but the effect size is opposite (e.g., GD with #SPR, GIN with T_{Bm}), and rarer cases where slopes reverse, yet the effect size aligns with the initial slope direction (e.g., RN with LoCa and LoCc).

Table 9: Trend direction of metrics per OSR before and after AI tool introduction. *Similar* denotes <20 % change in slope magnitude; *stronger/weaker* indicates ≥20 % change.

Direction	Maintained	Reversed
Similar	1 (1.2 %)	3 (3.6 %)
Weaker	18 (21.4 %)	18 (21.4 %)
Stronger	28 (33.3 %)	16 (19.0 %)

As a final step in analyzing the results, we re-examined the effect sizes in Figure 8 from the perspective of repositories rather than metrics, visualizing the distribution of effect sizes per repository in Table 10.

Table 10: Summary of OSR effect sizes, showing the number of positive ($\delta \geq 0.147$), negative ($\delta \leq -0.147$), and negligible ($|\delta| < 0.147$) effects; and average signed and absolute effect sizes.

	$\delta \geq 0.147$	$\delta \leq -0.147$	$\lvert\delta\rvert < 0.147$	Avg δ	Avg $\lvert\delta\rvert$
BS	2	10	2	-0.36	0.44
FA	8	4	2	0.23	0.49
GD	5	6	3	0.01	0.36
GIN	2	8	4	-0.16	0.21
RN	7	7	0	0.19	0.60
VSC	6	6	2	0.00	0.49
Total	30	41	13	-0.015	0.43

Signed average effect sizes mostly show small effect sizes in either direction, though more frequently positive. The repositories GD and VSC in particular show a near-perfect balance of effects, with respective averages of 0.01 and 0.00. Absolute averages cover all strengths, though the majority shows a medium effect. Still, we find the repositories BS and GIN standing out from the rest. GIN consistently shows some of the smallest effect sizes across metrics, and both its absolute and signed average effects differ notably from the others: the absolute due to it being the smallest, and the signed being one of only two negative averages. BS shows the other negative signed average, notably of medium strength, and frequently diverges from the other repositories in its metric-specific effects. As seen in Figure 8, BS often exhibits negative effects

in cases where the other four repositories (other than GIN) show medium to strong positive ones (e.g., #*C*, #PR, #SPR).

Lastly, we analyzed the previously applied grouping, where we categorized participants based on the dominant direction of effects with at least small magnitude, following the process outlined in section 3.3. If applied directly, the process would assign one repository to the positive, two to the negative, and the remaining to the mixed. However, since repositories include two additional metrics (14 total), the threshold for group assignment should be adjusted from 8 to 11 ($14 \times 0.75 = 10.5$). With this adjusted threshold, all but one repository fall into the mixed, with the exception of GIN being trendless. As such we can find some tendency toward the grouping, but their identification is not clear if we adjust the group criteria properly.

Summary: $RQ_{1.2}$ findings

The majority of metrics show at least a small effect (84.52 %), with a substantial portion being large. Pull request metrics exhibit the strongest effects, particularly in the number of pull requests and successful ones, which usually change collectively and often increase strongly. Additionally, their time to close decreases in nearly all cases, often with large effect sizes.
Commit-based metrics also show notable changes, with #*C* displaying both large positive and negative effects across repositories. In contrast, file-level metrics show the weakest effects, with one-third being negligible.
While effect directions are often balanced across metrics, absolute averages indicate consistent change. Trend directions are maintained in 56 % of slopes and reversed in 44 %, with most also changing in magnitude, showing no consistent directional trend.

4.4 $RQ_{1.3}$: How do the productivity effects of AI-assisted development compare between short-term industry and long-term open-source contexts?

$RQ_{1.3}$ focuses on the comparison of the results found for **$RQ_{1.1}$** and **$RQ_{1.2}$**, aiming to identify whether potential short-term impacts on productivity reoccur in different long-term settings, and thus identifying potential continuous trends or unique occurrences. To this end, we contrast the respective results

presented earlier, summarizing them in overview tables and figures to ease the comparisons without switching between sections.

Table 11 shows an overview of the distribution of effect strengths across the IR- and OSR-based data from their respective result sections. While medium effects remain largely stable, weak and no effects decrease, resulting in more large positive and negative effects. The ratio between the amount of negative and positive effect sizes remains relatively consistent, with negative effects still occurring more frequently.

Table 11: Percentage distribution of effect strengths by underlying data and direction of change.

Repository	Positive			Negative			No Effect
	Weak	Medium	Strong	Weak	Medium	Strong	
IR	16.1	7.1	11.3	24.4	8.3	10.7	22.0
OSR	10.7	4.8	20.2	22.6	8.3	17.9	15.5

These shifts are reflected in the average metrics shown in Figure 9 (derived from Figures 3 and 8): Previously, the IR data was dominated by no or small effects (signed and absolute, respectively), the averages now span all magnitudes (signed) or lean toward medium to large effects (absolute) for the OSR data. Interestingly, although the signed averages for IR-based metrics are small, they tend toward the negative, while OSR-based metrics show a positive direction – despite the higher frequency of negative effects.

In both cases, the strongest effects appear in pull request-based metrics, followed by commit-based ones. However, the increase in large effects from IR- to OSR-based data is concentrated in a few metrics. For commit-based metrics, the most notable change is the increase in commit frequency ($\#C$) by 0.28. Other metrics within this category show only minor changes (0.01–0.09), none of conspicuous magnitude.

All pull request-based metrics show increased absolute averages, with the time until a request is merged (T_{PRm}) changing the least (0.05). The averages reflect the pairwise movements between the total number of PRs and successful ones (#PR, #SPR) and the time measurements (T_{PRc}, T_{PRm}) for both IRs and OSRs. However, we observe an inversion in directionality and a weaker link between the time metrics. Specifically, #PR and #SPR shift from equal negative effects (IR) to equally stronger positive effects (OSR). In contrast, the time spent on PRs (T_{PRc} and T_{PRm}) shifts from an increase (IR) to a decrease. Furthermore, T_{PRc} shows an effect more than three times stronger for the OSRs than both its IR counterpart and the OSR-based T_{PRm}, which remains in the

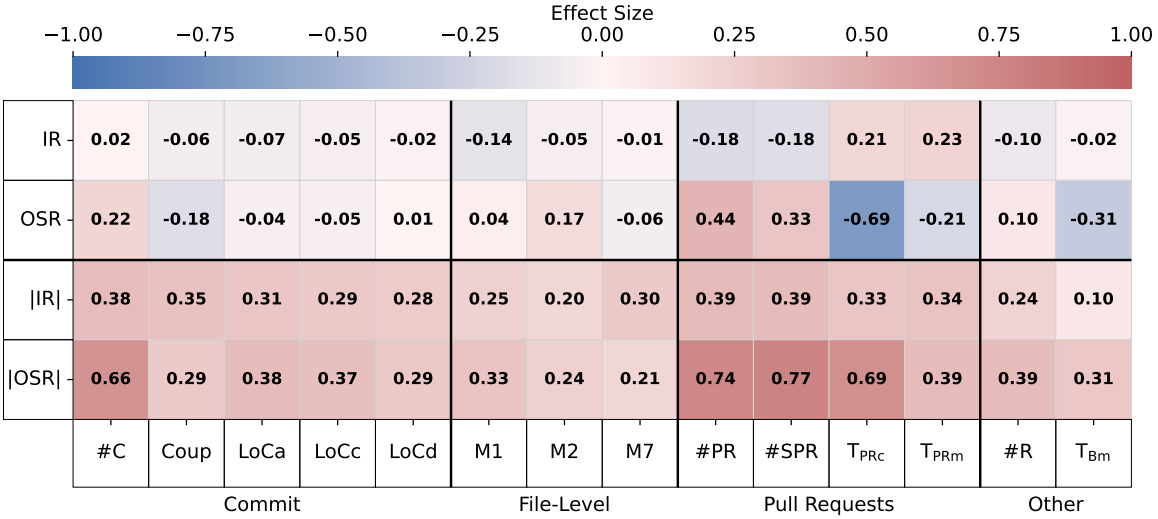


Figure 9: A comparison of the average metric changes from the IR and OSR data; including the signed and absolute averages from Figure 3 (IR and |IR|) and Figure 8 (OSR and |OSR|) separated in one overview.

same effect size category. For file-level metrics, the shifts remain in a minor magnitude with changes between 0.04 and 0.09.

The only metrics showing a decrease in absolute average magnitude are coupling (Coup) and one relative code churn metric (M7), which relates added/-modified LoC to deleted LoC. Aside from that, the file-level and other categories follow the same general trend of increased magnitude that has been viewed from different angles, but show no further notable changes.

Next, we analyzed the maintained and reversed slope directions shown in Table 7 and 9. We observe no notable directional shifts: similar and weaker trends decline regardless of direction, while stronger trends increase, mirroring the overall rise in large effects. Even special cases, such as two negative slopes with a positive effect size, occur at similar frequency, though not consistently across the same metrics.

Finally, we shift focus from the global view to the individual analysis of participants (IR) and repositories (OSR). Among participants, we observed clear groupings with consistently positive, negative, or mixed effect sizes, defined by at least 75 % of relevant metrics showing small or greater effects in the corresponding direction. This pattern does not hold for repositories: while some loosely fit into the categories if we use the same margins we used for the IR grouping, their categorization becomes entirely mixed once the margins are adjusted according to the 75 % of relevant metrics.

Summary: $RQ_{1.3}$ findings

Comparing our results based on shorter timeframe industry repository data with the longer timed open-source repository data, we find that effects generally become greater, both in terms of per-metric averages and changes in slope magnitudes before and after the AI tool's introduction. This rise in effect sizes affects pull request-based metrics the most, followed by commit-based ones. Specifically, we find the same paired movements between absolute (#PR and #SPR) and timed (T_{PRc} and T_{PRm}) pull request metrics, though their directionality is mirrored and the correlation between the timed metrics weakens, as T_{PRc} is notably larger than T_{PRm}.
The grouping of individuals, either participants (IR) or repositories (OSR), based on 75 % of relevant metrics showed some participants fitting each group, but lost its discriminative power for repositories, whose effects were more mixed and inconsistent.

Chapter 5

Discussion

To answer our **RQ$_1$** *How does AI-assisted development affect short-term developer productivity in industry repositories?*, we addressed three subquestions that explore the short-term effects within companies' private repositories, the long-term effects on open-source projects, and the implications of combining both results. To answer the subquestions, we formulated 5 hypotheses, continued with an exploratory analysis of the data. In this chapter, we discuss the implications of our results for **RQ$_1$**.

To structure our evaluation of the research questions, we begin by reviewing the results related to the initial five hypotheses on the use of LLM-based coding assistants in development. For each hypothesis, we examine findings for both **RQ$_{1.1}$** and **RQ$_{1.2}$**, then synthesize these insights to draw a final judgment per hypothesis in support of answering **RQ$_{1.3}$**. We then analyze additional findings on productivity impacts, verify them across contexts, and use them to formulate new, final hypotheses that inform our answer to **RQ$_1$**.

H_1: The use of an AI tool causes higher developer activity and development speed. This hypothesis is measured by absolute metrics (#C, #PR, #R). If true, we expect positive effect sizes, indicating increased activity after the AI tool's introduction.

For **RQ$_{1.1}$**, the results are mixed. While absolute averages suggest notable impact, this does not consistently reflect increased activity. Commit frequency increases and decreases with similar strength; pull requests more often decrease, with only three participants (P2, P5, P11) showing increases in both metrics. We therefore reject the hypothesis in this context.

In contrast, results for **RQ$_{1.2}$** largely support the idea that AI tools cause higher development activity and speed: both signed and absolute averages of #C and #PR show positive trends, with only one repository showing a consistent decrease. Release frequency (#R) drops in half the cases. Thus, the

hypothesis mostly holds for this setting, though not universally.

This results in a mixed picture for $\mathbf{RQ_{1.3}}$: While the findings for $\mathbf{RQ_{1.2}}$ align with prior work that generally reports increased development speed with AI tools [5, 13, 29, 31, 46, 47, 59, 61, 65], this was not replicated in the short-term, private-industry setting of $\mathbf{RQ_{1.1}}$. Instead, $\mathbf{RQ_{1.1}}$ reflects the less common view of decreased development speed [40], or findings suggesting variance of productivity impacts depending on the participant [31, 47].

This divergence could have multi-layered reasons, some of which are uncovered by the participants themselves within the expectations section of the questionnaire. Many participants, consistent with other studies [46, 47, 59, 65], reported low initial trust in the AI tool and its generated code. In open-ended responses, six participants explicitly mentioned concerns about code quality, particularly hallucinations, and the need to spend time verifying generated code. One participant, who was excluded from the general evaluation due to missing repository data, but completed the questionnaire nonetheless, also noted that learning how to prompt effectively would require additional effort, collaboration with colleagues, and time for the entire team.

While we do not find conclusive evidence supporting H_1 for $\mathbf{RQ_{1.1}}$, this may be due to the effect not manifesting within the short timeframe of our study. However, participants' self-reported concerns may help explain the short-term decline in development speed. These barriers are likely to diminish over time, as indicated by longer-term open-source results and prior work showing that trust in AI tools grows with use [31]. Based on these findings, we propose a new hypothesis in response to our overarching research question: *The use of an AI tool initially causes a decline or stagnation in developer activity and development speed, followed by an increase once users have adapted to its use.*

Other potential explanations for the observed increases, such as participants' experience levels, were considered but found less convincing based on our data. Given prior findings that AI tools benefit novice developers more [18, 23, 29, 70], we examined whether the three participants with positive effects differed in experience. These participants had 10-46 years of experience, which is not substantially different compared to the experience of other participants.

H_2: The use of an AI tool leads to higher code output with a greater task focus. We measure the expected increase in output using lines of code changed (LoCc) and greater task focus via coupling (Coup). If the hypothesis holds, we expect a positive effect size for LoCc (more added/removed LoC) and a negative effect size for Coup (fewer files per commit).

In the IR context, this effect only occurs for one participant (P12). For the rest, changes in both metrics vary in direction and strength, though they often

share the same sign.

While absolute averages suggest medium effects for Coup and small effects for LoCc, signed averages are near zero, indicating no consistent direction. This holds even when considering alternative code output metrics (LoCa, LoCd). We observe some tendency toward increased task focus, as 5 participants show medium to large negative effects for Coup, but a similar opposite trend exists for 3 other participants. The pattern for LoCc mirrors this, albeit with smaller effect sizes. Based on these mixed results, we reject H_2 in this context.

In the OSR data, the trend is similar but with a stronger signed negative average. However, this is influenced by one repository with a large negative effect size, skewing the overall result given the smaller sample (6 repositories vs. 14 participants). As the other patterns remain largely the same, we reject the hypothesis altogether. This suggests that while AI tools may support developers in solving tasks efficiently [3, 5, 12, 65, 69, 70], this does not directly lead to more focused commits.

However, this does not necessarily mean that developers are not *working* more focused. Prior work shows that commits often include changes from multiple tasks [16] – a practice that may persist regardless of AI assistance and could be influenced by company policies or team conventions. Such factors are not visible in commit-level metrics and are difficult to capture through abstract measurements. Still, given that focused commits are considered more maintainable, easier to review, and better aligned with best practices [2, 16], we continue to assume low coupling to be a desirable indicator of focused work.

Our results also show that the increase of produced code that has been found by other researchers [24] might not be the reality for every person or repository, especially considering only 1 OSR showed a large positive effect on the LoC metrics.

H_3: The use of an AI tool negatively impacts pull request handling time and reduces pull request success rates. For $\mathbf{RQ_{1.1}}$, we notice that total PRs (#PR) and successful ones (#SPR) generally move together. While medium-sized effects are present, they often decline in parallel, leaving the success ratio unchanged. Thus, we reject the hypothesis' assumption of reduced success rates.

In contrast, we do observe a nearly consistent increase in both T_{PRc} and T_{PRm}, with equal effect sizes and independent of the PR volume metrics. This supports the idea that developers initially need more time to verify AI-generated code, thus partially confirming the hypothesis in the short-term industry context.

The $\mathbf{RQ_{1.2}}$ results show the reverse: both #PR and #SPR increase proportionally. Meanwhile, both T_{PRc} and T_{PRm} tend to decrease, though not uniformly. In only two out of six cases are the effect sizes similar; others vary

significantly or even point in opposite directions. Notably, T_{PRc} is consistently negative across all cases and always more so than T_{PRm}.

This may either reflect improved developer ability to assess AI-generated code quickly or more effective use of AI tools during the review process. Both could in turn reduce the time the participants of the other setting expected to need for the task. Additionally, prior work suggests LLM-based coding assistants is helpful to understand new code [42, 50], which is necessary for quick pull request handling and could further explain the speed-up. As such, the hypothesis holds partly, though not in its entirety and is thus rejected.

To summarize for $\mathbf{RQ_{1.3}}$, these patterns suggest that while PR success rates remain stable across contexts, AI tool adoption initially slows PR handling, likely due to increased verification effort. Over time developers may adapt, leading to faster closure times. Merging continues to require more time, likely due to the persistent need for environment-level validation. We encapsulate this in a new hypothesis: *AI-assisted development initially slows development speed and increases pull request verification time due to the adaptation phase, but leads to improved efficiency as developers become more familiar with the tool.* The new hypothesis is similar to the one that emerged from discussing H_1, but focuses on a different metric to ensure both are individually tested and confirmed.

H_4: The use of an AI tool shifts the ratio of added to removed lines of code. The fourth hypothesis examines changes in the relative code churn rates (M1, M2, M7) without specifying the nature of these changes. It centers on shifts in development behavior reflected in the proportions of code generated, modified, deleted, and the overall volume rather than absolute numbers.

With nearly one-third of the participant-level effect sizes being negligible, relative code churn appears to be among the least impacted dimensions after introducing the AI tool. Both M1 and M2 exhibit mostly small, negative effects, suggesting that less code is being added, modified (M1), and deleted (M2) relative to file length. This may indicate reduced editing and refactoring activity overall, as it has been shown in the latest GitClear study [24], though the parallel decline in both additions and deletions weakens any strong directional interpretation.

M7 shows more variation, with effect sizes split evenly between positive and negative, and several participants exhibiting medium to large effects in either direction, and no apparent connection to the other two metrics. This implies that the balance between added/modified and deleted lines shifted for many participants – but in different ways. There is no dominant trend indicating whether additions or deletions increased relative to the other.

Overall, these findings suggest individual variation in how the AI tools influ-

enced editing behavior, rather than a consistent pattern across the participants. With regard to the hypothesis in $\mathbf{RQ_{1.1}}$, the results support a participant-dependent shift in the ratio of added to removed code, but the generally small effect sizes point to only minor directional changes and thus should be considered carefully.

Regarding $\mathbf{RQ_{1.2}}$, we observe a more balanced distribution of effect size directions across metrics M1, M2, and M7. Compared to $\mathbf{RQ_{1.1}}$, there are more medium to large effects, though no dominant trend emerges. This suggests that while the AI tool's impact is more pronounced in this dataset, it remains inconsistent across repositories. Interestingly, in 5 of the 6 repositories, M1 moves in the same direction as $\#C$, indicating that higher overall commit activity tends to align with increased relative additions and modifications per file. This may point to the participant-level behavioral patterns: those who commit more frequently also tend to generate proportionally more code churn.

Overall, these findings indicate that the AI tools do influence relative code churn, but the effect varies substantially between repositories. As such, the results are situated between claims of no changes [68] and of uniformly increased code churn [24], thus only hesitantly supporting the hypothesis using the data of $\mathbf{RQ_{1.2}}$.

Synthesizing this information for $\mathbf{RQ_{1.3}}$, we do not find specific trends in the $\mathbf{RQ_{1.1}}$ results which might develop into the broader $\mathbf{RQ_{1.2}}$ results. Given that 6 participants explicitly emphasized benefits in question answering and information discovery, rather than direct code generation, it might be that we find the AI tool impact to be more on a behavioral level, rather than fine-grained file level churn. Additionally, the observed differences between the two datasets may not be due to a specific learning effect over time, but instead reflect varying developer compositions across repositories. Some repositories may have contributors more inclined toward extensive editing, while others exhibit more conservative usage patterns – potentially driving the divergent trends.

Overall the exploration of the metrics used to test this hypothesis did not reveal a major trend. While we cannot completely reject this hypothesis, we can also neither judge it as true, nor update it more accurately, as there is no clear picture emerging.

H_5: The positive and negative effects of AI tools on productivity metrics balance each other out. To test this hypothesis, the signed averages for each of the metrics in the IR and OSR context is particularly informative. For $\mathbf{RQ_{1.1}}$, eight metrics show effects below the small threshold, suggesting that positive and negative changes either cancel each other out or are too minor to matter, resulting in overall balance.

PR-related metrics, however, show more directional effects. With total PRs and the successful PRs overall reducing, this may be concerning. Yet, since both decrease by the same margin, the success ratio remains stable, we view a decrease in activity, but not a lesser ratio of successful PRs. The increase in time spent per PR indicates a productivity loss, as this time could be used more effectively on other tasks [37]. Thus, the hypothesis holds partially on a per-participant level but is weakened by the negative impact on time-based metrics.

While it looks similar for $\mathbf{RQ_{1.2}}$, some trends emerge more distinctly. Activity metrics (#C, #PR, #R) mostly increase, though #R changes are too small to interpret. The increases in #C and #PR point to increased activity, which is a positive indicator for productivity, if supported by other metrics [44]. A decrease in Coup points to improved task focus and thus indicates that the activity rise can be considered positive. While both total and successful PRs increase, the smaller rise in successful PRs may hint at a slightly lower success ratio, but the effect is marginal and speculative. Lastly, all time-related metrics decrease, indicating less time being spent on PRs overall. This is generally viewed as positive, as it allows development work to proceed more continuously [37]. Additionally, the time until a branch is merged into the main branch (T_{Bm}) also declines, suggesting a more efficient integration process and contributing to faster delivery cycles [20].

Taken together, H_5 holds better in the short-term context, where effects balance out, while in the long-term open-source setting, some productivity effects tend to shift in a more positive direction. As such, we reject the hypothesis to answer our research question. Instead, we propose a new hypothesis based on our findings that better reflects the mixed, but leaning positive, findings from both related work and our results: *The positive and negative effects of AI tools on productivity metrics mostly balance each other out, developing a potential tendency toward positive effects over time.*

Exploratory observations. Across all hypotheses, there is another potential underlying explanation for the respective observations, which we did not discuss for each one individually. It may be that some developers differ in how strongly they are affected by AI assistance, resulting in them experiencing strong increasing effects, either negative or positive, from using AI tools, while others are more likely to experience decreasing effects. This would align with prior research showing varied outcomes depending on participants [31, 47].

To explore this possibility, we grouped participants based on their effect directions, resulting in 3 potential groups: Positive, negative, mixed, and a residual group that fit none. Using a threshold of 75 % of small effect sizes in either direction revealed that most of our participants fit in one of the groups.

We hypothesize that individual expectations, potentially influenced by attitude or personality traits, may explain the differing effects observed across hypotheses. While group sizes are small and thus interpretations are done carefully, this framing offers a promising lead for future research.

We investigated this idea by analyzing our questionnaire data based on the median responses of each group. One participant in $\mathbf{RQ_{1.1}}$, who did not fit into any group, stood out: they consistently rated themselves more experienced than peers from their team, from the company, and experts, expressed minimal concern about risks, expected high code quality, and reported both low intended usage of the tool and daily use across 7 of 14 scenarios. As the participant's low intended usage conflicts with their reported daily use across seven tasks, we assume they may have misread the usage item. Due to the study's pseudonymized design, we cannot confirm this directly with the participant. Based on this profile, we theorize that the tool may have less impact on (i) highly experienced participants, which is consistent with prior findings [18, 23, 29, 70], and (ii) those who apply the tool broadly across a wide variety of tasks rather than in focused ways. In the latter case, the lack of task focus may hinder the development of effective prompting strategies, a new skill required to learn, as was mentioned by four participants in our open-ended questions. Focusing on multiple tasks at once could lead to a shallower learning curve when using an AI tool and thus result in reduced benefits from the tool.

Participants in the positive group showed primarily positive effect sizes. They expressed greater concern about the AI tool than their peers, particularly regarding untrustworthy outputs and security risks. This caution may explain the increased time spent on pull requests, as they might be verifying AI-generated code more carefully. In contrast, participants in the negative group reported fewer concerns, which could lead to faster handling of pull requests, possibly explaining the quicker time metrics.

Lastly, participants showed differing expectations regarding which tasks they expected to use the AI tool for most frequently. The positive anticipated frequent to daily use for autocompletion, code testing, and code explanation, whereas the negative expected more use for code generation, autocompletion, and question answering. The mixed held a more neutral stance, expecting to use the tool occasionally across many tasks, with only autocompletion projected to be used more often.

These usage expectations may help explain differences in effect directions. Other research has shown that the preconceived expectations of using AI tools also constrained how they are used in early stages of an experiment [36]. Applying this to task-specific expectations, we assume participants focused their use of the tool on those anticipated areas at first. We thus hypothesize that the

type of task the tool is expected to be used for, and subsequently is actually used in, shapes its impact on specific metrics. This could be consistent with findings that AI tools are less effective for complex tasks [13, 46]. For example, use for code testing may primarily affect time or quality outcomes differently. Testing-related use might lead to faster merge times, as reviewers can more easily verify correctness when appropriate tests are already provided. Thus, how developers integrate the AI tool into their workflow likely influences both the direction and magnitude of its effects. However, we cannot confirm this theory directly, as we lack telemetry data on participants' exact tool usage, though it could provide an interesting lead for future research.

This perspective also raises new questions. In particular, we wonder whether developers' character traits shape their expectations, thereby influencing the productivity effects of AI tools. We also question how different usage areas affect specific metrics, and how these factors interact over time. While these theories are inspired by our findings, further research is needed to fully ground them.

All of this ties back to the repository productivity analyzed in $\mathbf{RQ_{1.2}}$, particularly the cross-setting comparison in $\mathbf{RQ_{1.3}}$. Here, patterns mirrored developer-level results, some reoccurred similarly, and many effects remained inconsistent or unidentifiable. Slope trends were inconclusive in both settings, and while effect sizes became stronger overall, no major distributional shifts were observed. This inconsistency may stem from the varying influence of developers belonging to one of the three groups within each repository. For instance, repository BS may be primarily influenced by negative participants, FA by positive participants, and GIN may lack a dominant group, thus possibly explaining why its observed metrics differ strongly from the other repositories. Since it is likely that not all developers within a repository follow the same group pattern, their effects may blend together. This could explain why the repository-level grouping was less effective than the individual-level analysis.

To summarize our exploratory findings, we propose three new hypotheses:

$\mathbf{H_{N1}}$: The extent to which LLM-based coding assistants impact developer productivity is influenced by individual developer characteristics, such as experience level and attitudes toward AI.

$\mathbf{H_{N2}}$: The specific tasks for which LLM-based coding assistants are used influence developer productivity differently.

$\mathbf{H_{N3}}$: Developers can be grouped and categorized based on the dominating effect direction of repository-based productivity metrics.

$\mathbf{H_{N4}}$: Repository productivity effects are largely decided by dominating groups of developers that exhibit similar AI tool usage patterns.

Metric collection. We are the first to employ this metric combination, aiming to provide an *objective* evaluation of the productivity impacts of AI tools.

The effect sizes have shown that these metrics may capture such productivity impacts, though they are not always easy to contextualize without additional human input. Still, some metrics correlate strongly, as reflected in their parallel movements. We saw this specifically for #PR with #SPR, T_{PRc} with T_{PRm}, and less dominantly for LoCc, LoCa, and LoCd.

Occasional exceptions emerged, though rather at the repository level. Despite their correlation, the metrics were chosen for their distinct perspectives. For example, while total and successful PRs often move together, a divergence between them could indicate increased development activity at the expense of quality. The absence of such divergence is encouraging.

However, in future research, especially with larger datasets, these correlated metrics should be examined carefully to ensure they do not disproportionately amplify certain effects in the analysis or distort the interpretation of overall productivity trends.

Considering the SPACE framework [21], which informed the design of our questionnaires, our metrics primarily address the dimensions of *Activity*, *Collaboration*, and *Efficiency*. Activity is reflected in the total (#) and LoC metrics. Efficiency is captured through time-based metrics, which represent delays that hinder developers from progressing with other tasks; total metrics can also reflect development speed and thus relate to efficiency. Collaboration is addressed through pull request metrics, as PRs typically involve multiple participants engaging in review and discussion.

As our analysis of the hypotheses, particularly the difficult interpretation of the results, has shown, SPACE is not complete without *Satisfaction* and *Performance*. The separate objective repository-level metrics, especially activity-based ones like LoC or #PR, have been criticized for being misleading or overly simplistic [26, 58]. As such, it is essential to emphasize that these metrics should not be used on their own and rather be viewed collectively to, in our case, assess the impact of AI tools across different productivity dimensions [27, 44].

It is not advisable to evaluate individual productivity using only the collected metrics. This is shown by 2 of the 14 not even working properly on an individual level, and especially the activity metrics being confusing if used on their own [44]. Instead they should be used in combination with metrics assessing S and P, as we did using the questionnaire.

However, if using the collection on its own, the metrics capture broader, repository-level impacts of AI tools on collaborative software development within a repository, where interpretation does not focus on individual people.

Answering RQ_1: How do LLM-based coding assistants influence productivity in real-world software projects?

Our analysis reveals that **RQ_1** requires a nuanced answer. While we had hoped to clarify the mixed picture presented in existing research on the productivity effects of LLM-based coding assistants, our findings remain inconclusive in that regard. We observed mixed effects across contexts. **$RQ_{1.1}$** and **$RQ_{1.2}$** presented differing patterns, and few consistent trends emerged in **$RQ_{1.3}$**. Rather than drawing a definitive conclusion, we aim to use these findings to motivate further research into *objective* productivity metrics that more reliably capture the productivity influence of LLM-based tools.
To this end, we summarize our contribution by proposing new hypothesis based on the findings of our original set (H_{OX}) and new (H_{NX}) hypotheses grounded in our exploratory findings. These hypotheses describe observed trends and potential explanations, but should be further tested on larger datasets across both private industry and open-source repositories. Our hypotheses include the following theories:

H_{O1}: The use of an AI tool initially causes a decline or stagnation in developer activity and development speed, followed by an increase once users have adapted to its use.

H_{O3}: AI-assisted development initially slows development speed and increases pull request verification time due to the adaptation phase, but leads to improved efficiency as developers become more familiar with the tool.

H_{O5}: The positive and negative effects of AI tools on productivity metrics mostly balance each other out, developing a potential tendency toward positive effects over time.

H_{N1}: The extent to which LLM-based coding assistants impact developer productivity is influenced by individual developer characteristics, such as experience level and attitudes toward AI.

H_{N2}: The specific tasks for which LLM-based coding assistants are expected to and used influence developer productivity differently.

H_{N3}: Developers can be grouped and categorized based on the dominating effect direction of repository-based productivity metrics.

H_{N4}: Repository productivity effects are largely decided by dominating groups of developers that exhibit similar AI tool usage patterns.

Chapter 6

Threats to Validity

6.1 Internal Validity

The *internal validity* describes whether the observed effects can be causally attributed to, in this case, the AI tools or the introduction thereof, or if they are a result of confounding factors.

While the time related analysis provides an interesting perspective, it also comes with the threat of time and its potential effect on the maturation of participants and developers in general. Maturation describes the natural evolution of developers' skill, habits, or other factors. This is particularly relevant in the open-source repository (OSR) dataset, where we cannot verify whether or when individual contributors began using an AI tool. As a result, observed changes might stem from developers' gradual improvement or adaptation unrelated to the tool itself. To mitigate this, we rely on the size of the dataset and the duration of the analysis window, under the assumption that individual variation averages out across thousands of commits and dozens of contributors.

For the industry repository (IR) data, we analyze a smaller time frame of around 6 months. While this time frame was not explicitly chosen to address this threat, it has an additional benefit. According to discussions with our company contact, palpable developer maturation, where personal skills, styles, and preferences change, requires a significant amount of time, that we likely did not cover. Nonetheless, external factors that we are not aware of, such as deliverable deadlines or release milestones, may have influenced development activity and overshadowed or amplified the effects of AI tool usage.

To mitigate these threats, we are careful to interpret small effect sizes, instead valuing stronger effects more. Rather than providing conclusive answers to our research question, we use our findings to identify indicative trends and formulate hypotheses for future research.

Furthermore, the choice of repositories may introduce a selection bias. The

results may vary between chosen repositories, as we have seen with GIN showing very different trends than the other repositories. We attempted to mitigate this by choosing repositories based on multiple factors (e.g., amount of stars, domain), creating a more balanced sample.

6.2 Ecological Validity

Ecological validity describes whether our findings are generalizable to real-world development contexts.

In our study, both industrial and open-source settings come with context-specific constraints. The IR data may be influenced by individual company policies, trainings, the work environment and other factors out of our control. Even OSRs, while publicly accessible, may be maintained primarily by corporate developers operating under similar constraints. While these factors may limit generalizability to entirely different contexts (e.g., independent or early-stage projects), they enhance ecological validity by grounding our observations in realistic, production-level settings. Both repository types are commonly used in software engineering research for this reason. Moreover, the IR data comes from a company that follows modern development practices, making it a realistic environment to study AI tool usage of developers.

To strengthen our ecological validity further, we intend to replicate the measurements with two more, priorly mentioned, companies' private repositories. These were not yet available to us, at the time of this thesis. Further open-source repositories are planned to be included as well. This ongoing analysis aims to further assess whether the observed patterns are consistent across varying organizational contexts.

6.3 Statistical Conclusion Validity

Statistical conclusion validity describes the degree to which conclusions drawn from our statistical analyses are justified and reliable.

An issue which our statistic conclusions have is the number of repositories and developers included in the analysis. With 14 (IR) developers and 6 OSRs, our analysis is based on a limited sample size, which may reduce the ability to detect smaller or more nuanced effects, or make individual trends seem bigger than they were. To mitigate this, we were careful in the interpretation of small subgroups and compared results across multiple contexts, allowing us to identify consistent trends rather than rely on isolated occurrences, thus strengthening the robustness of observed patterns.

The OSRs were carefully selected based on activity, maintenance status, and community engagement to ensure sufficient longitudinal data. Instead of treating our RQ as conclusively answered, we answered it with hypothesis that lay the groundwork for future more focused studies.

Given the use of 14 metrics across 14 developers and 6 OSRs, we acknowledge the potential for overinterpreting coincidental patterns due to multiple comparisons. To address this, we have been careful when interpreting singular small effect sizes, only focusing on combinations of re-occurring trends and higher values. We also explicitly focused on effect sizes to provide a more nuanced interpretation.

6.4 Construct Validity

Construct validity refers to whether this thesis' measures correctly reflect the intended constructs – in this case: productivity, programming experience and developer expectations regarding the usage of AI tools.

Productivity is an extremely complex concept, impacted by a wide range of factors, many of which might not even be capturable. Our metric set is not a one-size-fits-all truth and, as such, cannot accurately reflect productivity in its entirety in every context. Instead, the metrics highlight specific aspects of productivity and aim to do so by maximizing the perspectives available through completely abstract and high-level data. That enables their application even under strict privacy policies, where one seeks to gain a global grasp of the impacts of a tool. Nevertheless, we do not rule out further improvements to our collection of metrics in future research projects.

We do not attempt to measure productivity in its full conceptual breadth, but rather focus on a collection of objective and observable code-related metrics that have been informed by related work. We improve construct validity by measuring these 14 repository-based metrics together, and by further supplementing the analysis of developer productivity with their expectations when using the AI tool. This multi-leveled approach reduces construct underrepresentation and enhances interpretability.

Another risk we have is developers correctly judging their expectations previous to using the AI tool, at the time of the initial questionnaire. While the original plan was to collect the expectations right when the participants started using the tool, the reality was that they gained their licenses at different times, thus having access for varying timeframes. However, the questionnaire was voluntary and the answers were given conscientiously, as was noticeable by some extensive replies to open ended questions, as well as through the proper completion of such an extensive questionnaire. As such, we trust that

participants answered from the perspective they held at the start of their usage. Similarly, we have to assume participants did use an AI tool throughout the analyzed timeframe based on what we were told, as the companies could not share the telemetry data with us.

Chapter 7

Future Work

This work opens up multiple paths for future research, which we discuss in this chapter.

Character trait analysis. First, there is a considerable amount of individual developer data we collected in the initial questionnaire that, based on our results of expectations shaping productivity effects, could be used to deepen our observations and better judge the effects of LLM-based coding assistants on developer productivity. Specifically, we hypothesized in H_{N1} that individual expectations have a notable impact on how AI tools affect developers, and, if thought further, those expectations may be shaped by individual character traits. Our questionnaire data included detailed information about developers' personality traits. The questionnaire used was the validated BFI-S [63], a short-form version of the Big Five Personality Inventory [62].

A future direction is to analyze the participants' answers according to the official guidelines and then validate whether different personality dimensions correlate with different effects of AI tool usage. This could support more individualized assessments of tool impact across developer types.

Repeated measurements. Next, as we have discussed at multiple points in the threats to validity in chapter 6, the research could be strengthened by repeating the measurements with new data, specifically the repositories from the other two companies as well as more distributed OSRs. These additional repositories would allow testing whether the observed trends hold in different ecosystems, programming languages, and other factors.

Deeper view into repository data. With the available data, several additional perspectives could be explored. In this thesis, we focused on trends and changes, but future work might analyze the nature of these changes in more detail. For

instance, instead of relying on effect measures, one could compare absolute values or relative changes for each metric. However, this would introduce new challenges, such as the need to normalize metrics. For example, the number of commits in a repository like VS Code cannot be directly compared to those in a smaller project like Gin.

Interrupted Time Series analysis. Now that we can theorize that there are effects after introducing AI tools, we propose two more in depth time-based analyses.

First, we propose applying an Interrupted Time Series (ITS) analysis. ITS is appropriate when working with longitudinal data on an outcome variable (the time-bucketed metric values) and when a clearly defined intervention disrupts the time series, such as the introduction of the AI tool [22]. In essence, ITS uses the pre-AI-introduction data to fit a prediction model that estimates the expected behavior of the outcome variable (i.e., the metrics), had the intervention not occurred. This model is then extrapolated into the post-introduction period, allowing for a comparison between the predicted (counterfactual) and the actual observed data. The divergence between these two time series provides an estimate of the intervention's impact, including any level shifts or slope changes in the metrics over time. In our case, this could provide deeper insights into the changes that AI tools might introduce.

Time Pattern analysis. Another promising direction is the application of temporal pattern analysis to the repository metric data. While our current study evaluates aggregated trends before and after the AI tool's introduction, temporal pattern analysis allows us to investigate how these effects evolve over time, potentially capturing finer-grained dynamics such as sudden bursts, gradual shifts, or recurring cycles.

By applying time-series segmentation and change-point detection techniques, we may be able to identify when meaningful shifts in development behavior occur, beyond simple pre-/post-intervention comparisons [11], and assess whether these align with the introduction dates. This could offer a more nuanced understanding of the timeline once participants started using an AI tool and how the adoption process impacts their behavior, although we have yet to determine the exact techniques to apply [66].

Other pattern mining techniques could also be used to discover frequently occurring temporal motifs in developer activity, such as productivity spikes aligned with the tool's introduction, meetings where prompting techniques are discussed, sprint iterations, or stagnation phases [4, 14].

These insights could help distinguish the effects of AI tools from coincidental occurrences, and thus further strengthen the interpretability of the results.

Chapter 8

Lessons Learned

While this thesis has been a valuable learning experience overall, we use this chapter to reflect specifically on the lessons learned when conducting an industrial study involving external partners and large volumes of data. While a lot of the topics we will touch on can be condensed to a clear, concise and structured communication, we will now go into further detail.

First, a significant amount of invisible work goes into preparing such a collaboration with external organizations. In our case, we held multiple meetings with each company over a period of roughly three months. This, combined with email exchanges, waiting times, and rescheduled appointments, results in a time investment that must be factored into planning.

The study design was closely coordinated with our industry partners. While that has a lot of clear advantages, it also introduces a dependency on the partners to prioritize the non-profit research over other work commitments. Something that might not be possible in situations where there are other pressing deadlines. For example, when designing the questionnaire, this lead to long time delays when asking for selected feedback, while it delayed the process for the next contacts who also had other important matters.

From this, we derive **Learning $_1$**: *When input is requested but not critical, set a clear deadline in the initial communication. If the deadline passes without a response, proceed without the input; For example: "Please send your feedback by [specific date]"* – and actually stick to that deadline.

Another time-consuming challenge arises when essential information is needed beyond the collected data. Some metadata, such as the days when developers received access to the AI tools, cannot be collected directly by researchers, nor reliably recalled by participants. For example, participants typically cannot remember the exact date they gained access to LLM-based coding assistants, especially if it was several months ago.

To address this, we asked our company contacts to provide this information

after data collection. However, this approach proved problematic: we did not receive the necessary metadata in time, which led to excluding one company from the analysis, as we could not infer the missing dates. This leads us to **Learning 2**: *If essential metadata must be obtained from external sources, collect it before launching the main phase of the study.* Although delays are rarely intentional, it is frustrating to invest significant effort in data collection without being able to use the results.

When planning a study like this, it's important to recognize that some processes may be unfamiliar even to experienced external practitioners. In our case, this became clear during discussions around the pseudonymization of repository data. Since the technical and procedural aspects of this step were not widely known within the companies, it triggered extensive internal discussions, particularly between the involved practitioners and their data security departments.

These delays were not caused by unwillingness, but by uncertainty about how to proceed. The lack of a clear initial reference made it harder for external teams to align on a solution and slowed down the process of obtaining necessary approvals.

This brings us to **Learning 3**: *We strongly recommend preparing a written set of possible approaches in advance.* Offering concrete options gives external stakeholders a solid foundation to begin internal discussions, reduces miscommunications, and helps accelerate decision-making.

Finally, studies of this scope inevitably follow a defined timeline. While timelines are often discussed and verbally agreed upon, they might be disregarded when all parties are managing multiple responsibilities. Important dates and deliverables may be unintentionally overlooked due to competing priorities, holidays, product releases, or unforeseen absences.

This leads to **Learning 4**: *Establish a written timeline with clearly defined milestones and dates, and obtain explicit confirmation from all stakeholders. Encourage participants to verify the proposed schedule against internal calendars, including holidays, leave periods, and major organizational events, and plan for them to take time for this step* (remember **Learning 1**).

Once the timeline is finalized, it is advisable to operationalize it through scheduled reminders, such as automated emails, calendar invites, or messaging notifications. These mechanisms help maintain engagement, reduce the risk of delays stemming from overlooked commitments and take pressure off of the study-leading individuals to remember each separate party.

If the timeline cannot be followed as planned, there are two options: It can either be updated in agreement with all stakeholders, or continued with the remaining stakeholders – while those who cannot align with the current timeline follow a separate schedule and may rejoin or be included in later phases. The

first option is preferable under normal conditions without time pressure; the second is suitable when working toward a fixed deadline.

In our case, time delays led to two companies not being included in the study, as previously discussed. This also had a secondary impact, which is why we emphasize adjusting the approach accordingly: We collected and analyzed participants' expectations of using the AI tool, anticipating data from developers with shorter usage periods than those eventually included. To maintain a comparable baseline, we kept the expectation data. Had we committed earlier to excluding it, we could have focused solely on actual experiences, reducing threats to validity.

Chapter 9

Conclusion

In this master's thesis, we presented the results of a study investigating the effects of LLM-based coding assistants (AI tools) on objective, privacy-preserving productivity metrics from two perspectives. The metrics were derived from published research, aiming to quantify productivity using only repository data.

We first investigated the short-term effects on developers of private industry repositories, on a per-person level by analyzing the metrics for three months before, and three months after the developers gained access to an AI tool. This analysis was complemented by developer questionnaires, which gathered general background information, programming experience, and expectations regarding AI-assisted development. In the second phase, we collected long-term repository data spanning approximately 5.5 years from six open-source repositories. These were selected based on their popularity and domain relevance, and we applied the same productivity metrics as in the first phase.

Using the collected data, we calculated effect sizes to compare productivity before and after AI tool's introduction. For open-source repositories, we used the public release of GitHub Copilot as a reference point, and for the private industry repositories we used individual access dates provided by the company. Effect sizes provided a normalized view of the impacts on productivity. In addition, we computed slopes for pre- and post-periods, calculated absolute and signed averages, and aggregated relevant statistics to detect patterns and trends. These analyses helped estimate the influence of AI tools on repository-level productivity.

Our research was guided by three subquestions: One focused on short-term individual effects, another on long-term repository-wide effects, and a third aimed to compare both to understand how short-term adaptations translate into long-term impacts. To support this structure, we proposed five hypotheses, which we tested empirically.

Our findings show that understanding how coding assistants affect produc-

tivity in real-world repositories is a complex and nuanced problem. While our study contributes to theories that might reconcile conflicting findings in the literature and takes steps toward quantifying real-world impacts, a definite conclusion has not been found. We had to reject our five initial hypotheses, though we derived three new ones based on their insights.

Our data suggests that developers undergo a learning and adaptation phase with AI tools. This results in initial negative effects on development activity, speed, and pull request efficiency, followed by improvements once developers learn to use to the tools, e.g., by finding suitable prompts for their goals. We also observed that the positive and negative effects may eventually balance each other out, although with a general trend toward positive impacts, reflecting the ambivalence of current research. When contextualizing our findings with the developer questionnaires, we developed four new theories.

First, we hypothesize that individual developer attitudes influence how AI tools are adopted and how this adoption impacts repository-level behavior. Second, we assume that the specific tasks AI tools are expected to be applied to result in varying productivity effects. Third, we theorize that developers can be grouped based on the dominant direction of their productivity effects, and fourth, that the resulting majority group within a repository largely determines its overall productivity trend.

We summarize our results with seven new hypotheses that can serve as a foundation for future research. These are grounded in repository-level observations rather than self-reported data or controlled experiments, aiming to reorient research toward real-world impacts. Finally, we contribute a metric collection of 14 objective repository-based indicators, which can be applied to any version control system to establish a comparable basis of selected productivity dimensions for future studies.

With this work, we take an step toward unifying the conflicting narratives around the productivity impact of AI tools. Our research supports the development of objective, privacy-preserving metrics that reflect global productivity trends of entire repositories, keeping both company secrets and individual contributors out of the spotlight.

Appendix A

Questionnaire

The questionnaire was available in English and German. For space's sake, we will only share the three sections this study focused on in English.

Matrix questions are questions where, in our case, you can pick a single answer for each of the subquestions.

General

- As part of the study, you have been assigned an **ID** which represents your pseudonym. Please enter your **Study-ID** here: [Text]
- To assess whether impressions of the AI tool are especially influenced by different topics and task areas, we ask you to state in which project team(s) you primarily work: [Text]
- I / We work on software projects rather in... [Single Choice; answer options: Small teams (less than 4 developers), Medium-sized teams (4-7 developers), Large teams (more than 7 developers)]

Programming knowledge and experience

Please rate your programming experience using the following questions.

- What is your highest level of education? [single choice; answer options:No formal education, Lower secondary school leaving certificate (Hauptschulabschluss), Secondary school diploma (Realschulabschluss), Abitur, Completed training (abgeschlossene Ausbildung), University of applied sciences entrance qualification (Fachhochschulreife), Bachelor's degree, Masters degree, Promotion / Doctorate / PhD, Other]
- For how many years have you been actively programming on larger software projects (also privately)? [numeric]

- How many years have you been working as a professional developer in a company? [single choice; answer options: Less than a year, 1–3 years, 4–6 years, 7–10 years, Over 10 years]
- On a scale of 1 (=beginner) to 10 (=expert), how would you rate your programming experience? [Numeric 1-10]
- How do you rate your programming experience compared to... [matrix; answer options: Very inexperienced, Inexperienced, Neither inexperienced nor experienced, Experienced, Very experienced]
  - ... experts with 10 years of practical experience?
  - ... to your colleagues in the project team?
  - ... to your colleagues in the company?
  - The languages selected in the previous question
- What is/are your primary programming language/s? [Multiple choice; answer options: Java, Python, TypeScript/JavaScript, C#, C, C++, Other]
- How do you rate your skills in the following programming languages? [Matrix; answer options: Very bad, Bad, Neither good nor bad, Good, Very good]
  - ... experts with 10 years of practical experience?
  - ... to your colleagues in the project team?
  - ... to your colleagues in the company?
  - The languages selected in the previous question
- Which IDEs do you use? [Multiple choice; answer options: Jetbrains IDE (IntelliJ, PyCharm, ...), VSCode, Visual Code, Other]
- Which frameworks do you use [Multiple choice; answer options: React.js, Quarkus, Spring Boot, Angular, Other]

Expectations regarding the introduction of the AI tool

We want to capture your expectations at the time of the AI tools announcement. Think back to the time when the AI tool was first introduced in your company, e.g., while waiting for your license, and evaluate the following statements from that perspective.

- When the AI tool is introduced, I will, through its use, ... [Matrix; answer options: I completely disagree, I tend to disagree, I neither agree nor disagree, I tend to agree, I completely agree]
  - ...work more productively.
  - ...learn new things faster.
  - ...work more efficiently.
  - ...write better quality code.
  - ...organize my workload better.

  - ... work better in a team/with colleagues.
  - ... give answers I cannot trust.
  - ... give bad answers because it does not know my context.
  - ... pose a security risk because we do not have adequate policies.
  - ... cannot help me much because I'm not sufficiently prepared.
  - ... not be used much by me.
  - ... give me more work.
  - ... generate code.
  - ... have code completed automatically (autocompletion).
  - ... debug code.
  - ... test code or generate tests.
  - ... have code explained or analyzed.
  - ... identify requirements.
  - ... answers to questions (e.g. about programming languages or concepts).
  - ... automate recurring tasks.
  - ... document code.
  - ... structure, prioritize, and plan my tasks.
  - ... explore alternative designs and implementations.
  - ... support me with code commits and reviews.
  - ... generate test content and synthetic data.
  - ... create configuration files.

- When the AI tool is introduced, it will... [Matrix; answer options: I completely disagree, I tend to disagree, I neither agree nor disagree, I tend to agree, I completely agree]

  - ... work more productively.
  - ... learn new things faster.
  - ... work more efficiently.
  - ... write better quality code.
  - ... organize my workload better.
  - ... work better in a team/with colleagues.
  - ... give answers I cannot trust.
  - ... give bad answers because it does not know my context.
  - ... pose a security risk because we do not have adequate policies.
  - ... cannot help me much because I'm not sufficiently prepared.
  - ... not be used much by me.
  - ... give me more work.
  - ... generate code.
  - ... have code completed automatically (autocompletion).
  - ... debug code.
  - ... test code or generate tests.
  - ... have code explained or analyzed.
  - ... identify requirements.
  - ... answers to questions (e.g. about programming languages or concepts).
  - ... automate recurring tasks.
  - ... document code.

  - ... structure, prioritize, and plan my tasks.
  - ... explore alternative designs and implementations.
  - ... support me with code commits and reviews.
  - ... generate test content and synthetic data.
  - ... create configuration files.

- What other positive or negative effects do you expect from the AI tool? [Text]
- Once the AI tool is introduced, I will use it to... [Matrix; answer options: Never, Rarely, Sometimes, Often, Always]

  - ... work more productively.
  - ... learn new things faster.
  - ... work more efficiently.
  - ... write better quality code.
  - ... organize my workload better.
  - ... work better in a team/with colleagues.
  - ... give answers I cannot trust.
  - ... give bad answers because it does not know my context.
  - ... pose a security risk because we do not have adequate policies.
  - ... cannot help me much because I'm not sufficiently prepared.
  - ... not be used much by me.
  - ... give me more work.
  - ... generate code.
  - ... have code completed automatically (autocompletion).
  - ... debug code.
  - ... test code or generate tests.
  - ... have code explained or analyzed.
  - ... identify requirements.
  - ... answers to questions (e.g. about programming languages or concepts).
  - ... automate recurring tasks.
  - ... document code.
  - ... structure, prioritize, and plan my tasks.
  - ... explore alternative designs and implementations.
  - ... support me with code commits and reviews.
  - ... generate test content and synthetic data.
  - ... create configuration files.

- What **new** tasks or activities do you expect? [Text]
- For which tasks and activities would you **definitely not** use the AI tool? [Text]

Bibliography

[1] Idan Amit and Dror G. Feitelson. Which refactoring reduces bug rate? In *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*, 2019.

[2] Idan Amit and Dror G. Feitelson. The corrective commit probability code quality metric. *CoRR*, abs/2007.10912, 2020.

[3] Shraddha Barke, Michael B. James, and Nadia Polikarpova. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages*, 7, 2023.

[4] Dharmesh Bhalodiya, Jaydeep Tadhani, and Rajesh Davda. Mining recurring patterns in time series. *International Journal of Computer Applications*, 178, 2019.

[5] Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. Taking flight with copilot: Early insights and opportunities of ai-powered pair-programming tools. *ACM Queue*, 20, 2022.

[6] Joseph D. Blackburn, Gary D. Scudder, and Luk N. Van Wassenhove. Improving speed and productivity of software development: A global survey of software developers. *IEEE Transactions on Software Engineering*, 22, 1996.

[7] Markus Borg, Dave Hewett, Donald Graham, Noric Couderc, Emma Söderberg, Luke Church, and Dave Farley. Does co-development with AI assistants lead to more maintainable code? A registered report. *CoRR*, abs/2408.10758, 2024.

[8] Sayan Chatterjee, Ching Louis Liu, Gareth Rowland, and Tim Hogarth. The impact of AI tool on engineering at ANZ bank an emperical study on github copilot within coporate environment. *CoRR*, abs/2402.05636, 2024.

[9] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé 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, Joshua 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, 2021.

[10] Norman Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin*, 114, 1993.

[11] Thomas D. Cook and Sanjay Purushotham. A survey of methods for time series change point detection. *Knowledge and Information Systems*, 51, 2017.

[12] Nicole Davila, Igor Wiese, Igor Steinmacher, Lucas Lucio da Silva, Andre Kawamoto, Gilson Jose Peres Favaro, and Ingrid Nunes. An industry case study on adoption of ai-based programming assistants. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 2024)*, 2024.

[13] Begum Karaci Deniz, Chandra Gnanasambandam, Martin Harrysson, Alharith Hussin, and Shivam Srivastava. Unleashing developer productivity with generative ai. Technical report, McKinsey & Company, 2023.

[14] Jyo Deshmukh, Sayak Dey, Soonho Kong, Vinay Prabhu, and Mahesh Viswanathan. Specifying and detecting temporal patterns with shape expressions. *Software Tools for Technology Transfer*, 23, 2021.

[15] Anish Dhar. Ai coding assistants can be a huge help — just not where you think. *Built In*, March 2025.

[16] Martin Dias, Alberto Bacchelli, Georgios Gousios, Damien Cassou, and Stéphane Ducasse. Untangling fine-grained code changes. In *22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2015)*, 2015.

[17] Thomas Dohmke. Github copilot is generally available to all developers. GitHub Blog, June 2022.

[18] Thomas Dohmke, Marco Iansiti, and Greg Richards. Sea change in software development: Economic and productivity analysis of the ai-powered developer lifecycle, 2023. arXiv preprint.

[19] Sylvain Duranton. Are coders' jobs at risk? ai's impact on the future of programming. *Forbes*, April 2024.

[20] Nicole Forsgren, Jez Humble, and Gene Kim. *Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations*. IT Revolution Press, 2018. ISBN 1942788339.

[21] Nicole Forsgren, Margaret-Anne D. Storey, Chandra Shekhar Maddila, Thomas Zimmermann, Brian Houck, and Jenna L. Butler. The SPACE of developer productivity: There's more to it than you think. *ACM Queue*, 19, 2021.

[22] Federica Fusi and Jesse Lecy. Interrupted time series analysis, 2020. Course content from Public Economics for Public Policy (PE4PP).

[23] Haotong Ge and Yuemeng Wu. An empirical study of adoption of chatgpt for bug fixing among professional developers. *Innovation & Technology Advances*, 1, 2023.

[24] GitClear. Ai copilot code quality: Evaluating 2024's increased defect rate via code quality metrics. Technical report, GitClear, 2025.

[25] Ralph Gootee. Ai coding assistants are reshaping engineering — not replacing engineers. *The New Stack*, March 2025.

[26] Bill Harding. The four worst software metrics agitating developers in 2025. GitClear Blog, 2020.

[27] Ciera Jaspan and Caitlin Sadowski. *No Single Metric Captures Productivity*, pages 13–20. Apress open / Springer, 2019.

[28] Abin Joy, Senthilkumar Thangavelu, and Amalendu Jyotishi. Performance of github open-source software project: An empirical analysis. In *2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC)*, 2018.

[29] Ranim Khojah, Mazen Mohamad, Philipp Leitner, and Francisco Gomes de Oliveira Neto. Beyond code generation: An observational study of chatgpt usage in software engineering practice. *Proceedings of the ACM on Software Engineering*, 1, 2024.

[30] Amy J. Ko. Individual, team, organization, and market: Four lenses of productivity. In *Rethinking Productivity in Software Engineering*, pages 49–55. Apress open / Springer, 2019.

[31] Mohammad Amin Kuhail, Sujith Samuel Mathew, Ashraf Khalil, Jose Berengueres, and Syed Jawad Hussain Shah. "will i be replaced?" assessing chatgpt's effect on software development and programmer perceptions of ai tools. *Science of Computer Programming*, 235, 2024.

[32] M.J. Lawrence. Programming methodology, organizational environment, and programming productivity. *Journal of Systems and Software*, 2, 1981.

[33] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. A large-scale survey on the usability of AI programming assistants: Successes and challenges. In *Proceedings of the 46th International Conference on Software Engineering (ICSE 2024)*, 2024.

[34] Zhifang Liao, Xiaofei Qi, Yan Zhang, Xiaoping Fan, and Yun Zhou. How to evaluate the productivity of software ecosystem: A case study in github. *Scientific Programming*, 2020:8814247:1–8814247:13, 2020.

[35] Jalerson Lima, Christoph Treude, Fernando Marques Figueira Filho, and Uirá Kulesza. Assessing developer contribution with repository mining-based metrics. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE Computer Society, 2015.

[36] Jiqun Liu, Jamshed Karimnazarov, and Ryen W. White. Trapped by expectations: Functional fixedness in llm-enabled chat search. *CoRR*, abs/2504.02074, 2025.

[37] Chandra Shekhar Maddila, Sai Surya Upadrasta, Chetan Bansal, Nachiappan Nagappan, Georgios Gousios, and Arie van Deursen. Nudge: Accelerating overdue pull requests toward completion. *ACM Transactions on Software Engineering and Methodology*, 32, 2023.

[38] Alina Mailach, Dominik Gorgosch, Norbert Siegmund, and Janet Siegmund. "ok pal, we have to code that now": interaction patterns of programming beginners with a conversational chatbot. *Empirical Software Engineering*, 30, 2025.

[39] Boris Martinović and Robert Rozić. Perceived impact of ai-based tooling on software development code quality. *SN Computer Science*, 6, 2025.

[40] METR. Measuring the impact of early-2025 ai on experienced open-source developer productivity, 2025.

[41] N. Nagappan and T. Ball. Use of relative code churn measures to predict system defect density. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 284–292, 2005. doi: 10.1109/ICSE.2005.1553571.

[42] Daye Nam, Andrew Macvean, Vincent J. Hellendoorn, Bogdan Vasilescu, and Brad A. Myers. Using an LLM to help with code understanding. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE 2024)*, 2024.

[43] John Naughton. Now you don't even need code to be a programmer. but you do still need expertise. *The Guardian*, March 2025.

[44] Edson Oliveira, Eduardo Fernandes, Igor Steinmacher, Marco Cristo, Tayana Conte, and Alessandro Garcia. Code and commit metrics of developer productivity: a study on team leaders perceptions. *Empirical Software Engineering*, 25, 2020.

[45] Gergely Orosz and Addy Osmani. How ai-assisted coding will change software engineering: Hard truths. *The Pragmatic Engineer*, January 2025.

[46] Stack Overflow. Stack overflow developer survey 2024. ai, 2024. URL `https://survey.stackoverflow.co/2024/ai`.

[47] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. The impact of AI on developer productivity: Evidence from github copilot, 2023.

[48] Gustavo Pinto, Cleidson R. B. de Souza, Thayssa A. da Rocha, Igor Steinmacher, Alberto de Souza, and Edward Monteiro. Developer experiences with a contextualized AI coding assistant: Usability, expectations, and outcomes. In Jane Cleland-Huang, Jan Bosch, Henry Muccini, and Grace A. Lewis, editors, *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI (CAIN 2024)*, 2024.

[49] Foyzur Rahman and Premkumar T. Devanbu. How, and why, process metrics are better. In David Notkin, Betty H. C. Cheng, and Klaus Pohl, editors, *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, 2013.

[50] K.H.A. Reefman. Using llms to aid developers with code comprehension in codebases, 2024.

[51] DevOps Research and Assessment (DORA). Accelerate state of devops report 2024, 2024. Accessed: 2025-05-20.

[52] Daniel Rodriguez, M. Sicilia, Elena Barriocanal, and Rachel Harrison. Empirical findings on team size and productivity in software development. *Journal of Systems and Software - JSS*, 85, 2012.

[53] Jeanine Romano, Jeffrey D. Kromrey, John Coraggio, and John Skowronek. Appropriate statistics for ordinal level data: Should we really be using t-test and cohen's d for evaluating group differences on the nsse and other surveys? In *Annual Meeting of the Florida Association of Institutional Research*, 2006.

[54] Grant Ross. Ai coding assistants wave goodbye to junior developers. *CIO*, September 2024.

[55] Samaneh Saadat, Olivia B. Newton, Gita Sukthankar, and Stephen M. Fiore. Analyzing the productivity of github teams based on formation phase activity. In *2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, 2020.

[56] Ken Schwaber and Jeff Sutherland. The 2020 scrum guide™, 2020.

[57] Harsh Mukeshkumar Shah, Qurram Zaheer Syed, Bharatwaaj Shankaranarayanan, Indranil Palit, Arshdeep Singh, Kavya Raval, Kishan Savaliya, and Tushar Sharma. Mining and fusing productivity metrics with code quality information at scale. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2023.

[58] Tushar Sharma and Diomidis Spinellis. Do we need improved code quality metrics? *CoRR*, abs/2012.12324, 2020.

[59] Jiho Shin, Clark Tang, Tahmineh Mohati, Maleknaz Nayebi, Song Wang, and Hadi Hemmati. Prompt engineering or fine-tuning: An empirical assessment of llms for code. In *22nd IEEE/ACM International Conference on Mining Software Repositories*, 2025.

[60] Janet Siegmund, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. Measuring and modeling programming experience. *Empirical Software Engineering*, 19, 2014.

[61] Illia Solohubov, Artur Moroz, Mariia Yu. Tiahunova, Halyna H. Kyrychek, and Stepan Skrupsky. Accelerating software development with ai: exploring the impact of chatgpt and github copilot. In *CTE 2023: 11th Workshop on Cloud Technologies in Education*, volume 3679, 2023.

[62] Christopher J. Soto and Oliver P. John. The next big five inventory (bfi-2): Developing and assessing a hierarchical model with 15 facets to enhance bandwidth, fidelity, and predictive power. *Journal of Personality and Social Psychology*, 113, 2017.

[63] Christopher J. Soto and Oliver P. John. Short and extra-short forms of the big five inventory–2: The bfi-2-s and bfi-2-xs. *Journal of Research in Personality*, 2017.

[64] Adam Tornhill and Markus Borg. Code red: the business impact of code quality - a quantitative study of 39 proprietary production codebases. In *TechDebt@ICSE*. ACM, 2022.

[65] Thiago S. Vaillant, Felipe Deveza de Almeida, Paulo Anselmo da Mota Silveira Neto, Cuiyun Gao, Jan Bosch, and Eduardo Santana de Almeida. Developers' perceptions on the impact of chatgpt in software development: A survey. *CoRR*, abs/2405.12195, 2024.

[66] Gerrit J. J. van den Burg and Christopher K. I. Williams. An evaluation of change point detection algorithms. *CoRR*, abs/2003.06222, 2020.

[67] Justin D. Weisz, Shraddha Vijay Kumar, Michael J. Muller, Karen-Ellen Browne, Arielle Goldberg, Katrin Ellice Heintze, and Shagun Bajpai. Examining the use and impact of an AI code assistant on developer productivity and experience in the enterprise. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA 2025)*, 2025.

[68] Tao Xiao, Youmei Fan, Fabio Calefato, Christoph Treude, Raula Gaikovina Kula, Hideaki Hata, and Sebastian Baltes. Self-admitted genai usage in open-source software, 2025. arXiv preprint.

[69] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. Productivity assessment of neural code completion. In *Proceedings of the*

6th ACM SIGPLAN International Symposium on Machine Programming, 2022.

[70] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. Measuring github copilot's impact on productivity. *Communications of the ACM*, 67, 2024.