jamk

How Effective are AI-powered Code Assistants in Enhancing Developer Productivity?

Thin Thu Thu Thaw

Bachelor's thesis
May 2025
Bachelor's Degree Programme in Information and Communications Technology

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Description

Thin Thu Thu Thaw

How effective are AI-Powered Code Assistants in Enhancing Developer Productivity?

Jyväskylä: Jamk University of Applied Sciences, May 2025, 60 Pages.

Degree Programme in Information and Communication Technology. Bachelor's thesis.

Permission for open access publication: Yes

Language of publication: English

Abstract

The introduction of artificial intelligence (AI) technologies into software development has transformed programming activities. AI-based coding assistants, such as GitHub Copilot, Tabnine, and Codeium, support developers by generating code and providing contextual recommendations in real time. This research analyses how these tools impact productivity for developers with different levels of skills and varying degrees of task complexity.

A mixed-methods design was adopted wherein quantitative data was collected from 100 developers through structured surveys, and qualitative data was gathered through 10 semi-structured interviews. The thesis studied the effect of four independent variables on the dependent variable of developer productivity: the quality of code suggestions, task complexity, tool user-friendliness, and the user's level of expertise. The mean scores of all constructs suggested strong, positive perceptions concerning the usefulness of AI tools, as captured through descriptive statistics.

Correlation analysis indicated all independent variables to have significant relationships with productivity; in particular, the quality of code suggestions had the highest correlation ($r = .798$). Subsequent analysis via multiple regression confirmed only the quality of suggestion to be a significant predictor of productivity ($\beta = 1.162$, $p < .001$). Suggestion usability, suggesting expert level, and task complexity while being controlled for each other exhibited varying or no significant impacts.

Thematic analysis pointed to user trust, efficiency, and learning as primary advantages, while focus on over-dependence, security, and ethics of AI coding raised other concerns. The results substantiated that AI-driven code assistants boost productivity when suggestions are precise and the integration of tools is seamless. In addition, thoughtful design and monitoring are necessary to mitigate addiction and sustain software standards.

Keywords

AI-powered code assistants, developer productivity, GitHub Copilot, Tabnine, Codeium, code suggestion quality, tool usability, software development, task complexity, AI in programming, automation, mixed-methods research.

Miscellaneous (Confidential information)

No Confidential information.

Contents

Figures

Tables

1 Introduction

1.1 Study Background

The accelerating pace of artificial intelligence (AI) technologies is triggering considerable changes within the software development sector. One of the sector's greatest innovations is the release of AI-enabled coding assistants, including GitHub Copilot, OpenAI Codex, Amazon's CodeWhisperer, Tabnine, and Codeium. These tools utilize large language models (LLMs), especially those with transformer-based architectures, such as GPT-3 and Codex, to provide on-the-fly context-aware code suggestions, streamline routine programming operations, and help with the debugging and documentation tasks. The most significant advantage that these tools offer is to minimize the mental and physical burdens and rote programming tasks to shifting cognitive workloads, automating design decisions while focusing on reasoning instead of programming (Tan et al., 2024; Pinto et al., 2024).

GitHub Copilot, created by GitHub and OpenAI, is one of the technologies that adopt AI in an almost industrial manner. It functions as an AI pair programmer by constructing whole functions, completing monotonous code, and suggesting lines based on the developer's work. It supports many languages and works with editors such as Visual Studio Code. As reported by Liang et al. (2024), Copilot has transformed the interaction with code as a developer from creation to an augmented construction process, with the assistance of AI tools. It is adept at responding to natural language queries and providing the relevant snippets of code, which makes it easier to improve the speed of work and provide gaps in documentation or syntax knowledge.

Some researchers have studied the effect that these tools have on developers and their productivity. Sherje (2024) reported that Copilot users, especially in full-stack development settings, completed coding tasks significantly faster—up to 45% faster during automation heavy tasks like building CRUD operations and authentication modules. Also, in a comparative evaluation of Copilot and Tabnine, Pandey et al. (2024) reported that Copilot users had increased throughput and lower compilation error rates. In their economic assessment of the AI-infused developer lifecycle, Dohmke et al. (2023) noted significant improvements in time-to-deployment, bug reduction, and overall smoothness of the workflow in AI assistant-augmented shifts.

Nevertheless, considering these optimistic signals, a significant gap continues to persist in relation to the theoretical and practical understanding of the effectiveness of these tools in different contextual areas of development, as well as with respect to technology users and stakeholders. Most prior investigations concentrate on anecdotal developer accounts or miniaturized controlled tasks. For example, as Tan et al. (2024) noted, Copilot's aids are useful in dealing well-defined and documented issues, but its usefulness in ambiguous or complex issues is variable at best. In addition, Liang et al. (2024) emphasize that user satisfaction often hinges on the intricacies of the task at hand, the programming language in use, and the experience level of the developer. Such limitations point to the need for use and trust focused systematic and cross-sectional assessment of AI coding tools that moves past a mere functional value approach toward considerations of usability, value, learnability, integration into organizational workflows, and trust.

One critical issue that arises from the literature is the problem of relying too heavily on AI-generated code, particularly for learners at the development stage. Research conducted by Szolderits (2025) and Pinto et al. (2024) indicates that novices are likely to accept codes proposed by AI tools without comprehensively verifying its logic. This could result in stagnated skill development, unmaintainable code, and the unintentional creation of security flaws. A qualitative study done by Klemmer et al. (2024) on the ethics and security surrounding the use of AI code assistants found that some developers felt compelled to sacrifice accuracy for speed, trusting the AI suggestions even on important parts of the software. These results underscore an AI development tool's need for a balanced integration of AI technology into working environments.

Organizational and enterprise-level integration adds another layer of consideration. As Ng et al. (2024) and Ajiga et al. (2024) noted, enterprises are utilizing AI coding assistants in the public and high-tech domains due to projected efficiency gains and faster project completion. Still, these implementations are often constrained by data privacy concerns, insufficient tool adaptation, and integration of AI into legacy systems. For example, Codium has been able to capture market share for the on-premise deployment and customizable AI model features which appeal to companies with sensitive data (Addagalla, 2025). Nevertheless, there is a lack of cross-case comparative analysis on the effectiveness of these tools in different organizational settings.

Alongside operational issues, the ethical and legal boundaries of AI-generated code are increasingly scrutinized. Of primary concern in the discourse surrounding the responsible application of AI in software engineering are code attribution, licensing infringements, and model bias. Shah and Trehan (2024) and Lai (2024) focus on educating users about the limitations of AI to instill proper design responsibilities amongst programmers, underscoring the need for clearer action by AI tools. Additionally, Veeramachaneni (2021) points out that while AI-enhanced automated refactoring improves maintainability, it may also obfuscate the rationale for design choices, complicating future audits and debugging.

Examined from a wider theoretical perspective, the employment of AI aids relates to models of Cognitive Load Theory, Situated Learning, and The Technology Acceptance Model (TAM). According to Cognitive Load Theory, these aids seek to mitigate automated task inefficiencies by offloading repetitive work, thereby enhancing a developer's ability to allocate more mental energy to creative problem-solving (Szolderits, 2025). The Situated Learning Theory justifies the reason why Copilot functions as a contextual tutor, facilitating on-the-spot learning during live coding sessions (Pinto et al., 2024). From the TAM perspective, Cheng et al. (2024) and Liang et al. (2024) argue that user trust, relevance, and seamless integration are critical for the technology's uptake. These models of theories guide the design of this research, which seeks to evaluate the performance of AI tools along with the outcomes and perceptions of the developers.

Considering the rapid advances in AI and its profoundly disruptive effect on the practices of software engineering, there is an observable gap in empirical research exploring the productivity impacts of different AI tools across various user levels, tasks, and coding contexts. This research fulfills that gap by analyzing GitHub Copilot, Tabnine, and Codeium in terms of developer productivity, trust, usability, learning, and perceived outcomes. Using a survey and interview mixed-methods design, this study deepens the conversation around software development and AI integration from both practical and academic perspectives.

1.2 Purpose and Motivation

Addressing the practical gaps in this research aims to assess the AI-powered code assistants, particularly GitHub Copilot, to determine its impact on advancing software developers' productivity across different contexts. With the proliferation of these AI tools in professional and educational

coding settings, we can no longer rely on stories to explain their impact. Empirical evidence is needed. Preliminary research shows promise, suggesting gains in coding speed, mental load, and automation of routine tasks. However, the conditions under which these benefits are achieved are not well defined (Efthimia et al., 2025; Shah & Trehan, 2024).

This investigation strives to include comprehensive interactions and perceptions such as interaction, perceived usefulness, user satisfaction, and learning alongside completing tasks and the errors made during to provide a more holistic measurement of productivity. It answers a pivotal concern in modern software engineering discourse: Do AI assistants function as real productivity augmenters, or do they pose new dependencies and risks that surpass their advantages? (Dohmke et al., 2023; Szolderits, 2025). Specifically, this study analyzes GitHub Copilot because of its widely adopted user base, deep integration with IDEs, and the fact that it was built on OpenAI's Codex model which was trained on a large dataset of public code repositories.

An increasing body of literature suggests that measuring developer productivity solely by quantitative metrics such as lines of code and commits is an oversimplification. Productivity, from this lens, also includes context-specific cognitive efficiencies like reduced decision fatigue, confident assessment of logic accuracy, rapid understanding of previously unseen coding structures, and sustained flow state while coding (Pinto et al., 2024; Ng et al., 2024). As Kolusu (2024) elaborates, AI-assisted productivity is a function of how well the AI integrates into established practices and aligns with the developers' cognitive rhythm. This explains why this study attempts to broaden the scope of performance metrics to encompass usability and cognitive offloading as well as human-tool interaction.

Another motivating factor is the growing private sector and institutional investment in AI-assisted development. Major companies and public institutions alongside broad agile startup groups are deploying Copilot and Tabnine in production workflows with the hope of amplifying output while preserving or improving code quality. Ng et al. (2024) observed that with resource constraints, the public sector has begun using digital services engineering to integrate AI for project delivery at faster timelines. Ajiga et al. (2024) also observed that high-tech enterprises leveraged AI assistants to aid in the onboarding of distributed development teams to reduce onboarding tempo for junior

developers. Hamza et al. (2024) further note that scaled agile teams have come to view these instruments as critical enablers in the control of sprint velocity and backlog reduction. With all these above deployments, there is a lack of focus on the unresolved concerns of AI bias, tool transparency, reproducibility, and the control of code quality which this study seeks to address.

One of the most prominent reasons for utilizing AI-powered code assistants stems from their educational and instructive capabilities. Many people have claimed that GitHub Copilot and other similar tools serve as digital mentors that can help beginner programmers with syntax, functions, and general programming best practices. These tools can facilitate experiential learning, confidence-affirming programming in learners, and frustration alleviation by providing real-time feedback and contextual suggestions (Liang et al., 2024; Lai, 2024). However, the promise from a pedagogical perspective does come with risks. Prominent AI systems' suggestions might foster lack of understanding, stunted problem identification and resolution abilities, or mindless acceptance of flawed code or insecure solutions due to dependency. For this reason, the current study involves both beginner and expert participants to determine the impact of AI assistants on fundamental learning and software craftsmanship while examining whether they reinforce or disrupt these processes.

The opportunity to contribute towards resolving theoretical and empirical issues within the existing research literature is a gap that fundamentally motivates me. Most existing literature on productivity focuses on user satisfaction output metrics; however, very few provide comprehensive models that integrate theoretical, measurable performance outcomes like suggestion-quality of code, task complexity, user expertise, and usability of the tools. This research aims to fill the gap by creating and validating a conceptual framework which incorporates these interdependent variables. Developer perceptions, particularly trust and usability, as noted by Tan et al. (2024) and Cheng et al. (2024), have been shown to significantly influence the impact and adoption of AI tools for their functionality. I build on such ideas to examine how these perceptions relate to actual quantifiable outcomes of performance.

From a theoretical perspective, this research is linked to models like Cognitive Load Theory, Technology Acceptance Model (TAM), and Situated Learning Theory. With regard to Cognitive Load Theory, tools like Copilot may reduce extraneous load by performing command-level operations with syntax, so that developers only need to think at the design, logical, and structural level

(Szolderits, 2025). TAM focuses on adoption driven by perceptions of usefulness and ease of use, while Situated Learning Theory claims that Copilot provides contextualized, just-in-time learning that is tailored to programming tasks (Pinto et al., 2024; Liang et al., 2024). The combination of these theoretical principles constitutes an interpretive framework for the study that is rich enough to understand its findings and describe concrete steps for action.

There is also an ethical reason underlining this research. As questions of authorship, accountability, and software liability grow to be AI-written code, those problems emerge into sharper focus. Authors may, sometimes without a wish to do so, insert biased or unlicensed code into their code bases, which puts their company or themselves at risk legally, or of a bad reputation (Klemmer et al., 2024; Veeramachaneni, 2021). Also, tools based on public data can propagate bad old insecure coding styles if proper vetting is not done. This study addresses the gap on how users perceive and respond to such risks and hence contributes to the ongoing discourse on ethical implications of generative AI in software engineering.

Lastly, this research stems from the center of a practical problem, hence its purpose. The results of this study are expected to aid in making decisions by expounding insights that will guide software engineers, project leaders, educators, and even tool designers in knowing when and how to properly use AI code assistants. This includes decision making as to whether to deploy Copilot or Codeium at a certain company as well as their understanding of how junior developers would interact with AI during their onboarding phase. The results are aimed to impact the actual deployment of the tools, training programs and development policies directly.

The focus and rationale of this particular study derive simultaneously from practical business needs, scholarly investigation, and moral obligation. With respect to software development tools powered by artificial intelligence, this research seeks to increase understanding of their effectiveness and the ways in which they are accepted, trusted and incorporated into contemporary software engineering through a complete assessment of GitHub Copilot's capabilities in improving software developer productivity. It aims to confirm that assistance derived from AI technology is an enhancement of human capabilities instead of a diminisher of essential programming skills.

1.3 Thesis Structure

This thesis is divided into nine chapters, each of which aims to develop a comprehensive understanding of the impact of AI-powered code assistants, especially GitHub Copilot, on developer productivity in a sequential manner. This follows the research process from conceptual framing to field work and practical contributions.

Introduction (Chapter 1) outlines the context of AI in software development, the purpose and motivation for the study, as well as the broader significance of AI-assisted coding tools which helps in evaluating productivity. Usability, productivity, and AI-enhanced programming are some of the key terms that are framed for the subsequent inquiry. Chapter 2: Research Questions and Objectives presents the main research question along with its supporting sub-questions which are focused on task completion time, tool juxtaposition, limitations, and developer attitudes. This chapter is dedicated to establishing the core aspects of the investigation, providing focus and intent by defining limitations and scope. Chapter 3: In this case, both qualitative interviews and quantitative surveys were utilized in the study's mixed-methods research design outlined in Research Methodology. The chapter presents a literature-based case study comparison across GitHub Copilot, Tabnine, and Codeium, and describes the testing environment, data collection, and comparison framework, along with the ethics steps taken. It notes the safeguards implemented and how reliability and validity are upheld throughout the research process. In Chapter 4: Theoretical Framework explores the impact of AI on software engineering and covers the functionalities and underlying theories of AI-driven coding tools. It also conceptualizes developer output as a multidimensional metric. This chapter also examines Cognitive Load Theory, the Technology Acceptance Model (TAM), and Situated Learning Theory to explain the independent variables in relation to the outcome of the study. In Chapter 5: Practical Implementation and Results, the author provides the results from the empirical research phase of the study, which include practical evaluation and feedback from programmers on GitHub Copilot, Tabnine, and Codeium. Each subsection provides an evaluation of the assistants' performance in coding, efficiency in software development, execution speed in completing tasks, and includes statistical analysis and visual representations. Chapter 6: Discussion gives a critical interpretation of the findings in the context of research literature and history. It identifies the most important findings of AI coding tools, their strengths and weaknesses, and their implications for future software development. This chapter also analyzes the impact of these tools on developers, instructors, and institutions in a more general context. Chapter 7: Conclusion and

Recommendations synthesizes the primary outcomes of the research and outlines responsibilities associated with AI code assistants, including best practices of working with these tools. Suggested practical measures highlight the need to protect the identified risks while taking advantage of the tools. This section concludes with reflections on the dynamically changing interaction between software developers and intelligent machines. Chapter 8: References consolidates all the scholarly materials drawn on throughout the thesis and presents them in APA 7th style, maintaining academic thoroughness and integrity. Chapter 9: Appendices include additional materials and raw outputs, such as survey instruments, interview guides, tables, and data, that bolster the analyses in the previous chapters.

2 Research Questions and Objectives

2.1 Main Research Question

The main research question in this thesis is: How effective are AI-powered code assistants in improving developer productivity?. The aim of this question is to explore the practical impacts of AI assistant tools on coding speed, accuracy and developer satisfaction.

To support this main question, several sub-questions were developed to explore the specific aspects of the topic. These include: (1) Do these assistants reduce the time required for full-stack development tasks?, (2) How do different tools (Copilot, Tabnine, Codium) compare in terms of accuracy, efficiency and speed?, (3) What are the common limitations or risks while using AI-assisted coding tools?, (4) How do developers perceive the integration of these tools in their workflows?

2.2 Limitations and Scope

Even with a wide-ranging focus, this study accepts and addresses some of the limitations in scope to maintain rigor and feasibility. A striking constraint is the selection of tools. The study considers GitHub Copilot, Tabnine, and Codeium as representative AI-assisted code writing tools, but these do not constitute the full array of proprietary or experimental tools which are considered novel and putative, under development at this time. This decision stems from tool-level development, domain availability, and recent developer-centric popularity surveys (Liang et al., 2024; Tan et al., 2024). Another constraint is the specificity of the task context. The assessment is focused on full-

stack development for user tasks such as API integration, user interface (UI) rendering, and database interaction, as these present a reasonable blend of elaborate and algorithmic work where AI-generated prompts can be objectively considered. Domains such as embedded systems, data science workflows, or software engineering that is highly niche and specialized (like firmware for medical devices) are left out in order to preserve the integrity and precision of the results.

Another scope constraint is the diversity of participants. The research sample comprises software developers divided into three tiers: novice (0-2 years), intermediate (3-5 years), and advanced (6+) years. While there is an attempt to include participants from different geographic or organizational perspectives, these factors may impact the degree to which the sample is representative of the entire developer population. This may influence the extent to which the results about users' perceptions are culturally or infrastructurally generalizable. Furthermore, the study does not intend to address the social, legal, or economic consequences of AI-generated code in detail, such as the licensing of codes, intellectual property, or theories concerning economic displacement. These issues, while important, relate to a wider scope than simply measuring productivity and analyzing the user experience. Such readers are invited to consult Glushkova (2023) and Veeramachaneni (2021), who discuss systematized implications in more detail.

Lastly, this research is limited to the scope of a controlled experiment supplemented by surveys, suggesting that the results pertain only to the tasks, tools, and metrics within the predetermined boundaries of the design. This captures the essence of real-world complexities such as the evolving scope within a project, team collaboration dynamics, or the iterative testing cycles at production levels. Nonetheless, the study permits these shifts in focus primarily to the relevant AI coding aides and standard full-stack development which provides depth of analysis and clarity in key AI coding assistant contexts. Additionally, full interview transcripts were not retained due to technical constraints. However, detailed notes taken during and after the interviews were used to ensure thematic richness and accurate interpretation. This study provides actionable evidence-based recommendations relevant to software developers, AI tool creators, and technology strategy leaders about the real quantifiable productivity enhancements and the desirable and undesirable consequences of integrating AI assistants into the software development lifecycle.

3 Research Methodology

3.1 Research Design

This research utilizes a mixed-methods approach composed of both qualitative and quantitative techniques to evaluate the effectiveness of AI-based code assistants: GitHub Copilot, Tabnine, and Codeium on developers' productivity. The productivity of a software developer has observable, quantifiable components like time and resource allocation (e.g., time saved, code errors) alongside experiential elements (e.g., trust, integration into their workflows). This study's quantitative aspect focuses on capturing specific task-related data through a structured survey with 100 software developers, measuring and assessing task difficulty, receipt of code suggestions, tool usability, and productivity. The qualitative aspect entails 10 semi-structured interviews with a purposive sample of developers gained from the survey respondents in order to understand their experiences, opinions, and these tools in practical coding contexts.

These two methods combined enable a complete triangulated analysis allowing the study to confirm the trends identified in the survey with more detail gained from the interviews. This approach aligns with Pinto et al. (2024) and Cheng et al. (2024) AI assisted developmental studies which focus on blending empirical analysis with qualitative insight. In addition, this study is informed by human-computer interaction and software engineering research which recognize the need to evaluate developer tooling from a quantitative and behavioral perspective.

3.2 Literature Review and Case Study

In conducting this study, the reviewed literature relating to the adoption of AI in software engineering was the starting point. In particular, Sherje (2024) and Szolderits (2025) believed that GitHub Copilot and similar applications could boost productivity if certain conditions are met, especially by alleviating coding tasks through suggestion-heavy real-time text generation and by providing prompt syntax suggestions. However, most research tends to have a narrow focus on a single tool or developer cohort. This study fills that void through its design, which utilizes a multi-tool case study approach.

This research incorporates a case study approach to evaluate three AI-enabled code assistants: GitHub Copilot, Tabnine, and Codeium. Each tool is evaluated under the same API implementation, user interface, and database operation tasks common in full-stack development. Study participants are assigned to one of the three tools for observation, and their workflows along with feedback are collected. This approach strengthens ecological validity and real-world relevance, which Copilot included in the additional design dimension. The framework follows the proposed structure by Pandey et al. (2024) and Tan et al. (2024) focusing on AI tools for programming and their comparison with each other.

3.3 Operationalization Table

To clarify the structure in the measurement of key concepts, the following table describes the operationalization of all the study variables. Each construct is linked to specific indicators and corresponding measurement scales based on the sources. This framework forms the basis for both the quantitative survey and the subsequent analysis phases of the study.

Table 1. Operationalization of Variables

Variable	Measurement	Indicators	Scale	Source
Developer Productivity	Time saved, quality of output	Time to complete task, code correctness, number of iterations, satisfaction, task throughput	5-point Likert	Sherje (2024); Dohmke et al. (2023)
Code Suggestion Quality	Relevance and usefulness of suggestions	Context accuracy, syntax correctness, function completeness, repetition rate, logical fit	5-point Likert	Tan et al. (2024); Szolderits (2025)
Task Complexity	Developer-rated difficulty	Logic depth, external libraries used, unfamiliarity with domain, conditional branching, UI logic handling	5-point Likert	Shah & Trehan (2024); Kalava (2024)
Tool Usability	Ease of tool interaction and integration	Integration in IDE, error correction ease, latency, interface clarity, customization options	5-point Likert	Pinto et al. (2024); Ng et al. (2024)
User Expertise Level	Developer background and experience	Years of experience, familiarity with AI tools, language proficiency, prior Copilot use, task confidence	5-point Likert	Liang et al. (2024); Cheng et al. (2024)

3.4 Data Collection and Comparison Criteria

This study's data collection was based on two sources of information: a quantitative survey and a series of qualitative interviews. From the survey which included demographic questions and 25 Likert scale items tied with the operationalized variables, 100 complete responses were collected

from participants who, by screening, had at least some level of familiarity with AI-based development tools. The survey was shared as a document on software developing communities on GitHub, Reddit, and LinkedIn. The responses were created to seem structured collection for both comparability and statistical assessment with SPSS.

At the same time, a targeted group of survey respondents was invited for 10 semi-structured interviews. The participants were chosen based on their level of experience and skill in dealing with tools across many domains. Each interview, which was conducted with permission and lasted between thirty to forty-five minutes, was recorded. The interview aimed at focusing on trust perceptions in the outputs of the AI tools, integration into workflows, experienced outcomes of learning, and ethics to foster a comprehensive understanding.

While using GitHub Copilot, Tabnine, or Codeium, participants were all given the same coding problems and their performance was assessed for time taken, task success rate, and number of manual corrections needed to be done. This part of the experiment was based on Corso et al. (2024) and Upadhyaya (2024). The combination of objective metrics and feedback provided by participants captured subjectively is useful in supporting multifaceted verification and makes the findings more reliable.

Table 2. Data Collection Process

Activity	Start Date	End Date	Description
Literature Review	March 4, 2025	March 11, 2025	Reviewed relevant academic literature to frame research objectives and tools.
Survey Design and Piloting	March 12, 2025	March 18, 2025	Developed survey items based on operationalized variables; conducted pilot tests.
Quantitative Survey Distribution	March 19, 2025	April 5, 2025	Online distribution to 100 developers; data collection completed via Google Forms.
Interview Participant Selection	April 6, 2025	April 7, 2025	Selected 10 participants from survey pool using purposive sampling.
Semi-Structured Interviews	April 8, 2025	April 20, 2025	Conducted interviews via Zoom and took notes for analysis.
SPSS Statistical Analysis	April 21, 2025	April 30, 2025	Performed descriptive statistics, reliability testing, and correlation analysis.
Thematic Analysis of Interviews	April 21, 2025	May 5, 2025	Coded transcripts manually and identified emergent themes.
Integration of Findings	May 6, 2025	May 15, 2025	Triangulated quantitative and qualitative data for interpretation.

3.5 Procedures for Data Analyses

As for quantitative data, it was analyzed using SPSS. With the Help of SPSS, summary statistics were calculated to describe basic demographics of the sample as well as summarize trends in responses. Each variable set was analyzed for internal consistency by calculating Cronbach's Alpha which for this case, a value over 0.7 would indicate the data set is reliable. This proves the reliability of the indicator sets. After that, correlation analysis was conducted to test the relationship between productivity of a developer and other independent factors such as the quality of code suggestions, the level of task complexity, usability of the tools, and level of expertise of the user. These relationships were further explored using multiple linear regression analysis to measure the productivity affected in relation to all these other factors. The regression analysis answered the research questions about the factors that are most critical in enhancing performance in coding.

Alongside, the qualitative interview information was transcribed and analyzed theoretically according to Braun and Clarke's (2006) six-step framework. It included initial coding, theme identification, theme refinement, and final synthesis. Trust in AI tools, learning reinforcement, reliance concerns, usability: frustrations, integration: advantages, and many more emerged as collective themes across narratives from participants. This thematic analysis further enriched the understanding of user behavior and experiences, which tend to get overlooked in quantitative data.

3.6 Accuracy and Ethics

The reliability and ethical accuracy of the study was closely maintained. Reliability in the quantitative portion was confirmed with Cronbach Alpha tests, and survey item revisions followed a pilot study with ten participants. In the qualitative portion, inter-rater reliability was upheld through coding comparisons from two separate researchers to ensure consistency in theme extraction.

Ethical issues were not overlooked in any aspect of the study. Each participant was provided with a consent document detailing the objectives of the study, the voluntariness of their involvement, as well as the data collection and storage procedures. No sensitive data was gathered. Interview audio recordings were not made due to technical constraints, and transcripts summaries were prepared from detailed notes. Institutional approval from the ethics committee was obtained, and the research complied with institutional norms regarding ethics in research. Primary participants were also interrogated on ethical issues for AI technologies, as described in Klemmer et al. (2024) and Solanke (2023), focusing on fears like code plagiarism, dependency on technology, or bias from AI content generation.

This academic research utilizes a mixed-method approach wherein a quantitative survey provided primary empirical data, and qualitative interviews served a secondary, complementary role. Measuring productivity perceptions via AI-powered coding assistants within a baseline framework among a larger sample of software developers was the primary goal. These results provided insights regarding efficiency, usability, and code quality.

The survey results were the primary driver for the contextualized semi-structured interviews aimed at validating the emerging trends. Pinto et al. (2024) and Cheng et al. (2024) recommend these types of interviews and argue that multi-method triangulation strengthens the credibility

and interpretation of the results. During the interviews, users provided detailed reasoning for their preferences as to why certain developers favored GitHub Copilot over Codeium and tool usability changes concerning the project and development phase.

The survey's results revealed trends that were then elaborated upon in the interviews. This integration not only framed a more robust internal validity of the study but allowed the study findings to be interpreted with breadth and depth. Thus, this blended methodological approach fosters a comprehensive appraisal of the research problem combining quantitative precision and a narrative interpretation.

3.7 Justification for Survey and Interview Questions

Literature on the subject AI-powered code assistants and their effect on developer productivity have been synthesized, forming a conceptual framework which guided the creation of surveys and interviews for this study. Specifically, the survey sought to measure four distinct factors: suggestion quality, task complexity, tool usability, user expertise and overall developer productivity in relation to AI-powered code assistants. These factors were derived from Pinto et al.'s (2024), Tan et al.'s (2024), and Liang et al.'s (2024) studies, which highlight the complex interplay of AI coding assistants in development workflows.

To maintain instrument rigor, each construct was measured with five discrete Likert scale items as aligned with the operationalization plan created during the study design phase. This turns subjective developer experiences into quantifiable data in a structured, scalable, and analyzable way. For example, interacting with AI suggested code of high accuracy, relevant to the task at hand, and adaptable to evolving project demands assessed perception of suggestion quality while responsiveness and ease of workflow integration measured tool usability.

The semi-structured interviews aimed to examine the developers' experiences, challenges, and expectations concerning the constructs of interest to reveal the participants' understanding of the constructs at a deeper level. As noted by Shah and Trehan (2024) and Szolderits (2025), this qualitative aspect was important to capture subtle details that might remain hidden beneath the surface when using closed-ended survey questions. For that reason, the interviews were guided by the core themes of participants' reasoning about trust and autonomy, their trust and autonomy

ecosystem, and their perceptions of AI assistants over extended timeframes. This two-pronged strategy provided a better understanding of the main research question.

4 Theoretical Framework

4.1 AI in Software Development

In the context of software engineering, Artificial Intelligence (AI) is an automation resource that significantly enhances the individual processes in the software engineering lifecycle. AI generative models are employed at every phase of software development from high-level specifications to design and requirements gathering, where work is analyzed, code is optimized, repaired, and the necessary algorithms for AI aid are identified. Applications of CI, TDD, and agile methodologies most obviously need speed and dependability at the same time, which AI can provide. With AI, there is a change in the entire paradigm of software engineering which is recalibrated in “AI-informed” or "AI-founded", which according to Glushkova (2023), shifts the focus from human factors to where creativity is not restricted.

AI impacts software development in many ways. Some machine learning algorithms are created specifically to identify coding anomalies, automate unit testing, and scrutinize security flaws. Tools like DeepCode and CodeGuru give recommendations based on AI after scanning numerous codes, assisting developers to write efficient and secure application codes. The more recent surge in the use of Large Models, and especially transformer models like GPT-3 and Codex, has significantly increased the scopes of AI. These models understand natural language and supply contextual properly structured code snippets in multiple programming languages (Dohmke et al., 2023). Hence, AI is moving from a purely helpful role to a co-creator role in software development.

The evolution of AI within the principles of software engineering parallels emerging concepts in intelligent systems and CSCW, which advocate for the assimilation, not the alienation, of human capabilities. Kalava (2024) contends that the development of workflow-supporting AI requires consideration of a user's context, human attendance, and their psychological well-being. To put it succinctly, AI systems need to help and interact with developers in the execution of their roles while

offering broad access to tailored guidance for numerous programming tasks. From this perspective, AI acts as a cognitive collaborator proposing solutions, identifying emerging patterns, and adapting alongside user behaviors.

The productivity benefits of AI in development are important; however, there are still associated challenges. Ethical issues related to: transparency, accountability, and bias still persist. The rise of 'black-box' models that formulate code without explaining logically how it was derived, caused concern among both developers and institutions because it undermines frameworks of trust and sustainment of code maintenance (Klemmer et al., 2024). Furthermore, the incorporation of AI alters the patterns of team interactions and processes work scope such as ownership of code, communication, and error management. All these concerns provide justification for conducting practical studies on the real-world impact of AI in software development.

4.2 Assistants That Offer Code Suggestions Using AI

A notable change in AI and programming has come with the introduction of functional assistants such as GitHub Copilot, Tabnine, and Codeium. They utilize LLMs and in particular, OpenAI's Codex for powered Copilot to offer prompt tailored coding suggestions, advanced auto-completion, and continuous assistance within IDEs. Their primary purpose is to alleviate the cognitive load of programmers by automating mundane tasks and providing sophisticated development help (Tan et al., 2024).

Regarded as a revolutionary milestone in the field of AI-enabled programming tools, GitHub Copilot was released in unison with OpenAI. It has processed billions of lines of code and is capable of offering pointers ranging from simple functions to complex algorithms based on text or code snippets. Pinto et al. (2024) focused on the usability studies regarding GitHub Copilot and concluded that some developers found it useful in dealing with monotonous coding tasks or identifying problems for which well-established solutions were available. However, their findings suggested that some novice users tended to blindly accept suggestions made by Copilot, which underscores the need for informed supervision.

Also, Market entrants include Tab Nine, which is powered by custom-trained models of GPT-J, and Codeium, which is known for its emphasis on data privacy and security at an enterprise level.

These tools offer varying degrees of customizability, performance, and integration relative to the development framework and the needs of the company. Ajiga et al. (2024) argued that Codeium set itself apart by supporting on-premise deployment, thus catering to developers working with sensitive information. Simultaneously, Tab Nine users can collaboratively train their models, which improves coding assistance based on shared user practices.

The development of these assistants is grounded in the Situated Learning Theory as well as Cognitive Load Theory. Situated Learning Theory posits that learning occurs more effectively in the context where it will be applied. Therefore, AI-based assistants like Copilot provide situated learning by offering instant instruction through relevant coding suggestions during the coding dialogue (Szolderits, 2025). At the same time, Cognitive Load Theory, as proposed by Sweller in 1988, states that reducing unnecessary mental work enhances the function of our working memory. In this scenario, developers are conserving mental work guided by AI because there is no need to recall commands, look up pertinent information, or generate repetitive code fragments.

Despite this concern, the sociotechnical frameworks of the tools suggested still come with their own distinct issues. Perspective users' trust, perceived usefulness, along with ease of use form strong determinants of adoption according to Liang et al. (2024) concerning Technology Acceptance Model (TAM). Adoption of Copilot or Tab Nine by developers is likely to be underwhelming if the tool is seen as intrusive, improperly designed, or overly complicated to be seamlessly integrated into work processes, regardless of its functionalities. Moreover, within socio-technical systems contexts, value derived from the assistant's accuracy extends beyond performance; it includes adherence to team norms, workflow constructs, and respect for the individual developer's right to self-organize (Cheng et al., 2024).

Together with Lai (2024), Shah, and Trehan (2024) discuss ethical boundaries of code plagiarism alongside proprietary output safety and fostering mentally unhealthy coding behaviors. All of these issues warrant concern regarding the workflows, training, or general integration of AI assistants to the workplace. Ultimately, responsibility must be shared by the developers, who should position themselves as a secondary decision-maker in a system that functions primarily as an advisor instead of a final arbiter.

To summarize, these AI-assisted coding tools symbolize the deep intertwining of technology, intellect, and social collaboration. The use of these tools requires the system's technical functionalities alongside trust, collaboration, and ethical conduct, which shape the ideas that this research pursues regarding developer productivity.

4.3 Developer Productivity

In most cases, the concept of developer productivity has been elusive, difficult to measure, and strange to assess. It is well known that novel heuristics like counting lines of code or bugs fixed are far too simplistic in measuring complex activities of a developer, as they focus on sheer productivity volume. Current notions of developer productivity encompass efficiency, effectiveness, satisfaction, and morale, while measuring accuracy of the code, debugging time, task completion speed, and ability to maintain "flow" (Sherje, 2024). (Dohmke et al., 2023).

In this research, we examine developer productivity as a feature with multiple dimensions. Aspects such as completing a job to user satisfaction and the number of errors are both qualitative and quantitative. Furthermore, during a task, a developer's interest, user satisfaction, and mental strain are additional qualitative measures. The model proposed aligns with recent interpretations of productivity discussed in software engineering literature (Tan et al. 2024; Shah and Trehan 2024).

Sager and Wahab (2024) suggest one of the most effortless methods AI assistants increase productivity is by minimizing context shifts during work. To illustrate, suggestion retrieval does not necessitate browses to the internet or documentation; therefore, suggestions can be retrieved during workflow. Such an improvement aligns with flow theory put forth by Csikszentmihalyi (1990), which describes uninterrupted focus as a facilitator of substantial productivity and satisfaction.

Through the use of AI tools, developers can allocate minimal time to routine matters such as the creation of boilerplate code and API integrations (Kolusu, 2024). Implementation times during basic CRUD functions can be up to 40% more efficient, although the overall effectiveness tends to drop with increasing task complexity (Pandey et al., 2024). This drop generally occurs because while the tool completes autonomous execution of fundamental components, it is less adept at scaling complex parts or intricate domain logic.

The use of Activity Theory enables the examination of the actions of the developer, the AI assistant, and the task environment in relation to productivity for the current study. Productivity is defined not only as individual effort, but also as derived from a balance between the tools employed, goals set, and the execution of tasks. When using AI assistants, well-designed tools, when integrated seamlessly into the developer's workflow, enable smooth goal attainment. Conversely, unhelpful or disruptive AI guidance becomes a source of friction.

The other aspect concerns how an AI developer assistant shapes productivity in relation to a developer's experience. As noted by Szolderits (2025) and Liang et al. (2024), AI assistants cater actively to beginning developers because they provide ongoing guidance and error checking. They often support streamlined workflows and even automate recurring tasks. Too much support, however, may hinder the acquisition of essential skills. Experienced developers, however, generally do not take full advantage of AI; they tend to use these tools strategically for routine work and prior to critical decisions.

Moreover, productivity must be viewed through the lens of the software's future quality. Glushkova (2023) argues that while AI has the potential to increase immediate productivity, automation, or neglect in the validation and understanding processes of the code generated, could have dire consequences in maintenance down the line. Thus, this research seeks to measure productivity along with factors related to quality assurance to avoid fast development cycles resulting in excessive technical debt, critical security weaknesses, or crucial vulnerabilities.

5 Data analysis

5.1 Quantitative Data Analysis

5.1.1 Demographic analysis

The demographic profile of the 100 participants in this study shows that the global developer community is well represented. From an age distribution perspective as illustrated in Figure 1 and detailed in Table 3, the most dominant group was 41-50 years old (26%), followed by above 50 (19%), between 21-30 (21%), and less than 20 years (18%). Respondents aged 31-40 accounted for 16%. This range depicts a combination of varying experience levels, including entry-level developers, students, and established professionals in the field. The older developer's presence indicates a

breaking down of barriers to the use of AI tools, even among older professionals who have historically relied on manual coding and shunned automation.

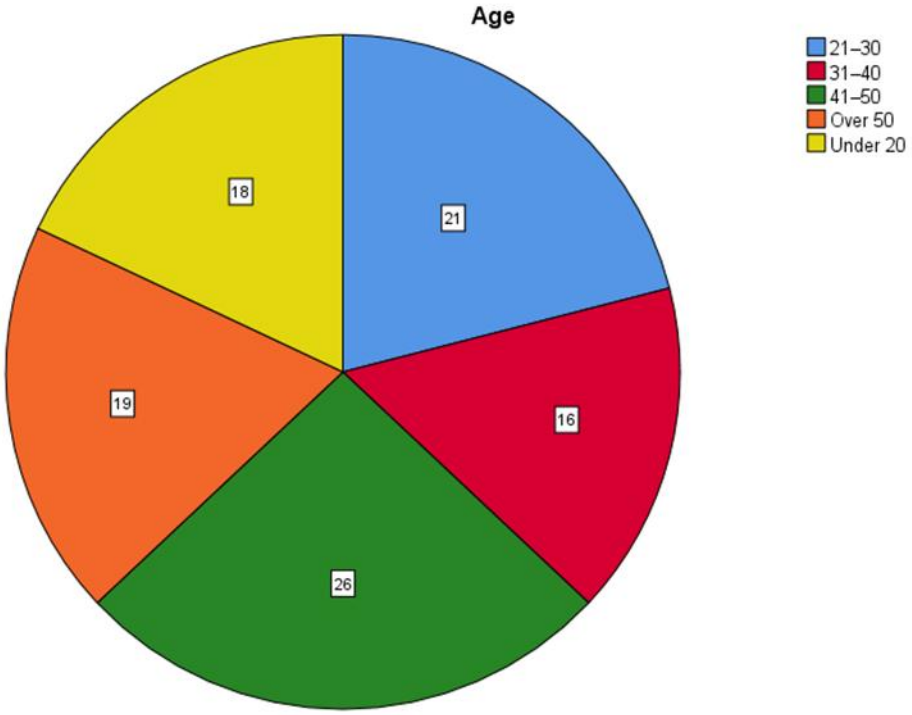

Figure 1. Age Distribution of Respondents

Table 3. Age Distribution of Respondents

Age					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	21–30	21	21.0	21.0	21.0
	31–40	16	16.0	16.0	37.0
	41–50	26	26.0	26.0	63.0
	Over 50	19	19.0	19.0	82.0
	Under 20	18	18.0	18.0	100.0
	Total	100	100.0	100.0	

In terms of gender distribution as illustrated in Figure 2 and detailed in Table 4, the balance was relatively maintained with 44% female and 56% male respondents. Although the technology industry has been known to be a male domain, this sample suggests that the balance is shifting towards greater female participation in AI-assisted development. Such a balance in representation is important for examining potential gender based differences in the perception or usage patterns of AI-powered code assistants.

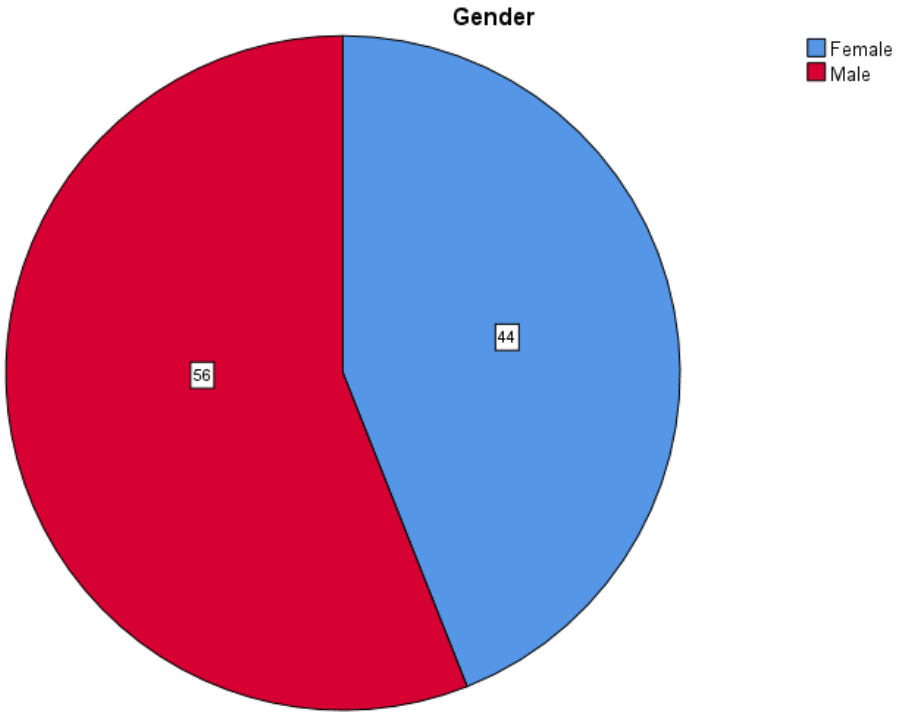

Figure 2. Gender Distribution of Respondents

Table 4. Gender Distribution of Respondents

Gender					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Female	44	44.0	44.0	44.0
	Male	56	56.0	56.0	100.0
	Total	100	100.0	100.0	

Another demographic factor that adds to diversity is the country of residence. According to Figure 3 and detailed information from Table 5, Sri Lanka contributed the highest proportion of participants, followed by the USA, UK, and India, with smaller contributions from other countries: Sri Lanka (28%), USA (24%), UK (23%), India (16%), and Others (9%). This adds to the cross-cultural credibility of the study by ensuring that the findings are not too influenced by developmental socioocultural environments or national technological infrastructure frameworks.

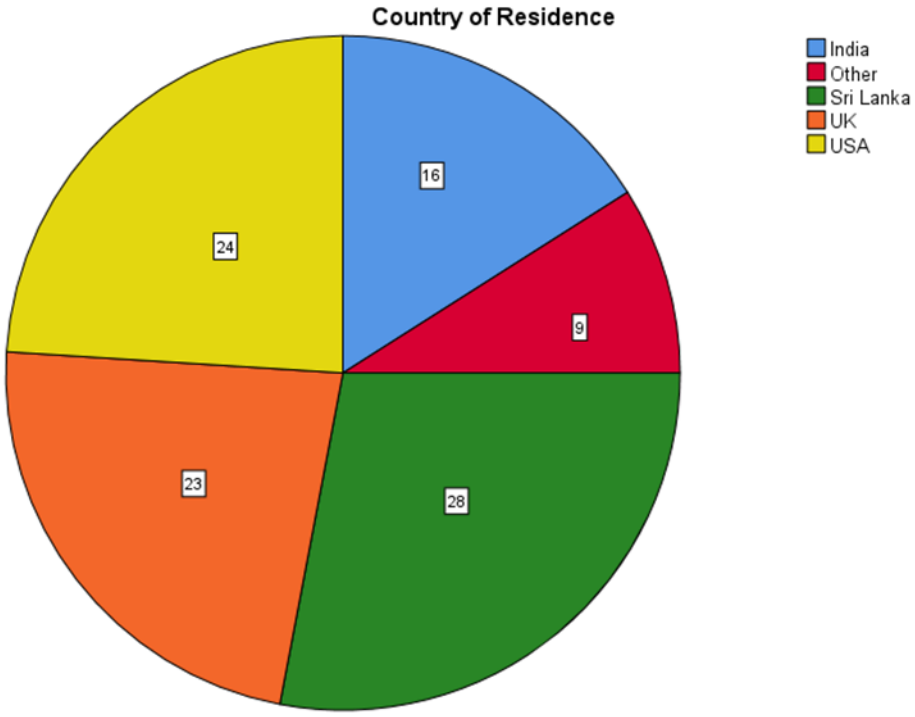

Figure 3. Country of Residence of Respondents

Table 5. Country of Residence of Respondents

Country of Residence					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	India	16	16.0	16.0	16.0
	Other	9	9.0	9.0	25.0
	Sri Lanka	28	28.0	28.0	53.0
	UK	23	23.0	23.0	76.0
	USA	24	24.0	24.0	100.0
	Total	100	100.0	100.0	

Regarding programming experience as illustrated in Figure 4, respondents were evenly distributed. A sizable portion of respondents, which was 23%, had under one year of experience, and an additional 20% had between one and two years. Meanwhile according to Table 6, 19% had 3–5 years, another 19% had 6–10 years, and 19% reported over 10 years of experience. This diverse composition allows for a study on the impact of AI tools on productivity at different career stages, starting from learners and junior developers to senior specialists.

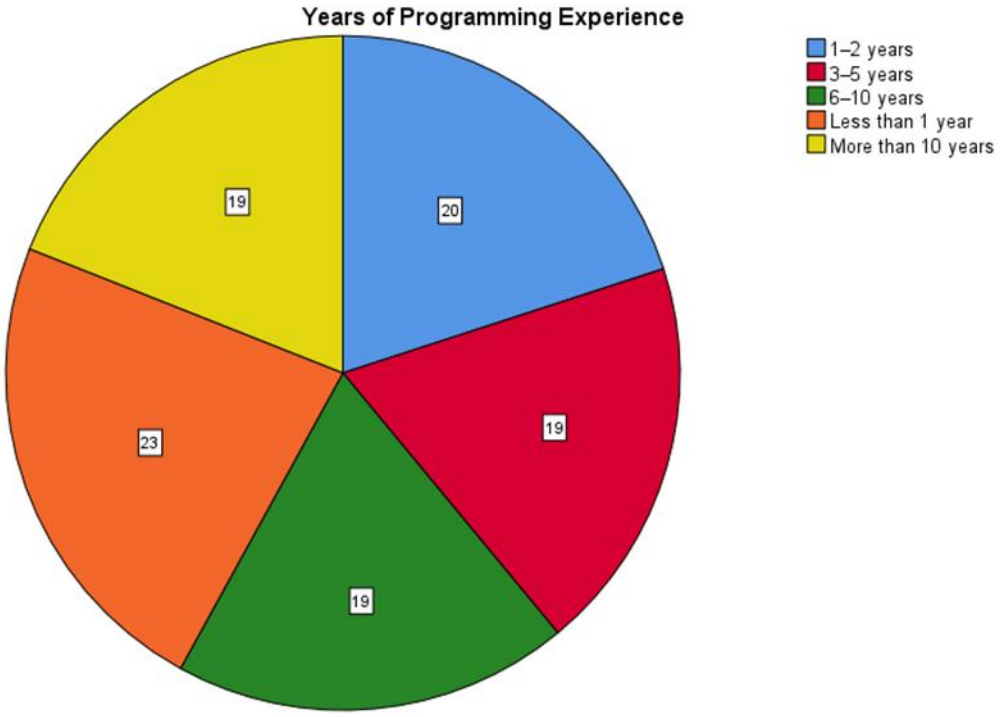

Figure 4. Programming Experience Levels of Participants

Table 6. Programming Experience Levels

Years of Programming Experience					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	1–2 years	20	20.0	20.0	20.0
	3–5 years	19	19.0	19.0	39.0
	6–10 years	19	19.0	19.0	58.0
	Less than 1 year	23	23.0	23.0	81.0
	More than 10 years	19	19.0	19.0	100.0
	Total	100	100.0	100.0	

Observing the professional roles in Figure 5, the respondents comprised a nearly even split between junior (22%), mid-level (14%), and senior developers (24%). In addition, Table 7 shows that students made up 15% of the sample, while team leads and architects represented 25%. This role composition improves the trustworthiness of the study since the varied expectations and uses of AI tools in actual development work enhance the applicability of the findings.

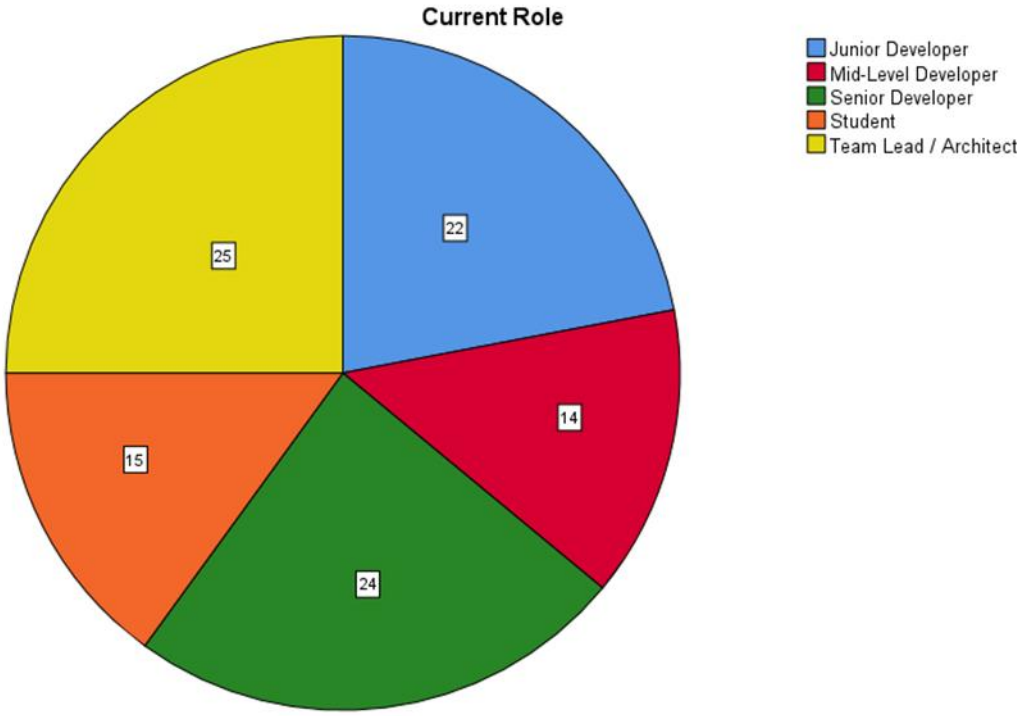

Figure 5. Current Professional Roles of Respondents

Table 7. Current Professional Roles

Current Role					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Junior Developer	22	22.0	22.0	22.0
	Mid-Level Developer	14	14.0	14.0	36.0
	Senior Developer	24	24.0	24.0	60.0
	Student	15	15.0	15.0	75.0
	Team Lead / Architect	25	25.0	25.0	100.0
	Total	100	100.0	100.0	

In the context of AI coding assistants as shown in Figure 6, GitHub Copilot had the highest usage rates with 31%, Tabnine followed with 26% and Codeium being only slightly lower at 24%. Table 8 also shows that a smaller portion of the population (19%) reports using other AI assisted coding tools. This distribution mirrors the growing popularity of Copilot while also suggesting that competing tools are starting to gain traction, particularly among those developers considering privacy, cost, or customization options.

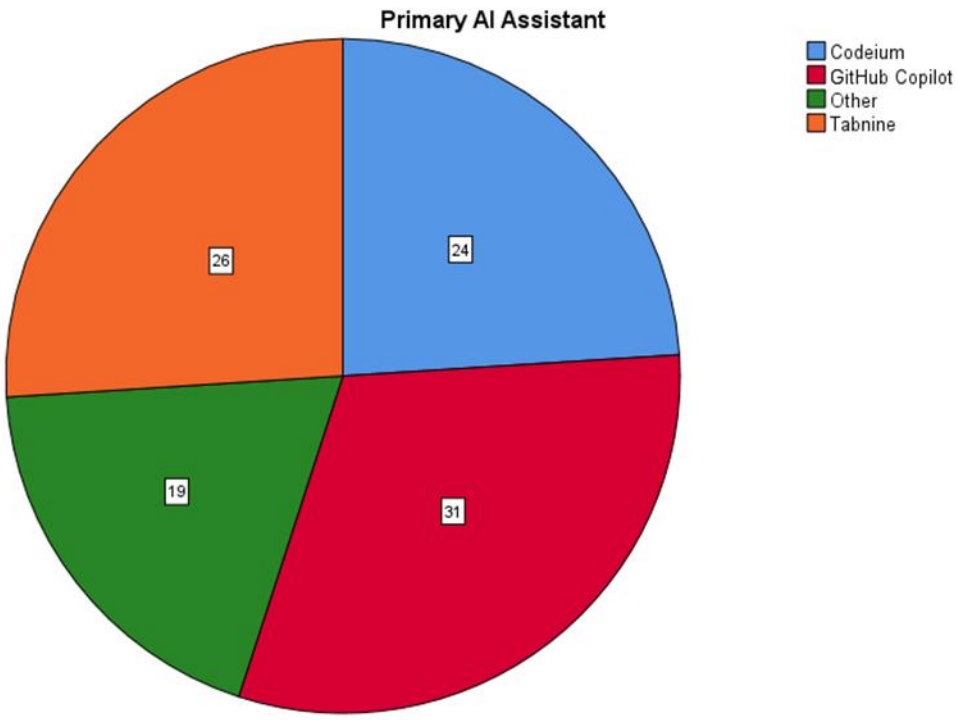

Figure 6. Primary AI Coding Assistant Used

Table 8. Primary AI Coding Assistant Used

Primary AI Assistant					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	Codeium	24	24.0	24.0	24.0
	GitHub Copilot	31	31.0	31.0	55.0
	Other	19	19.0	19.0	74.0
	Tabnine	26	26.0	26.0	100.0

Lastly in Figure 7, participants reported JavaScript (24%), Python (18%), C# (17%), Java (16%), and other languages (25%) as their primary programming dialects. Additionally, Table 9 represents the use of AI assistants within a broad range of syntactic environments and technological stacks, confirming the tools' adaptability and highlighting the need to assess their impact in a context specific to the languages' particularities.

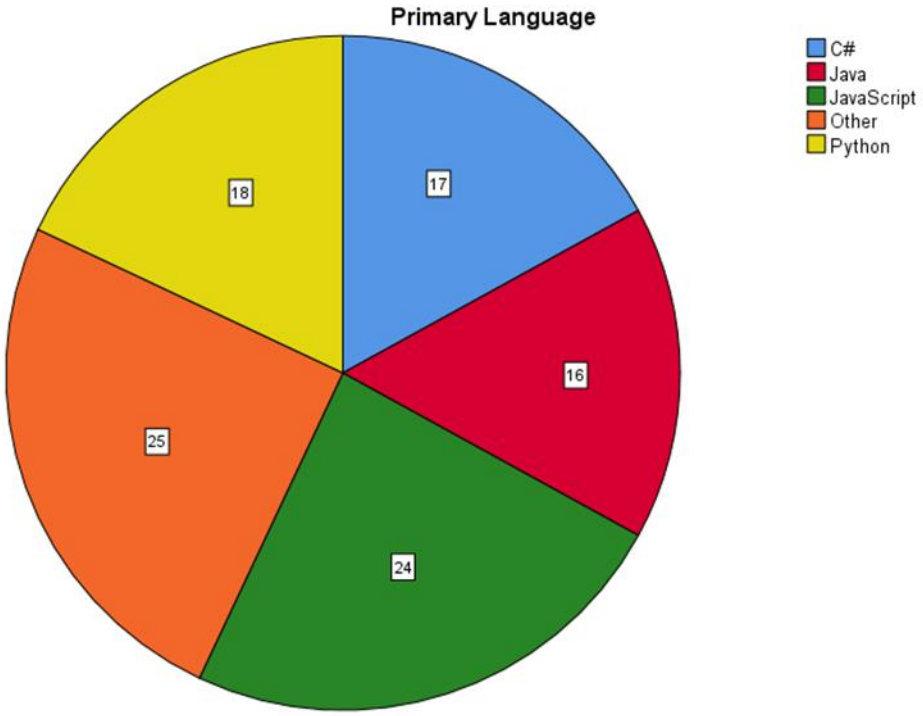

Figure 7. Primary Programming Languages Used

Table 9. Primary Programming Languages Used

Primary Language					
		Frequency	Percent	Valid Percent	Cumulative Percent
Valid	C#	17	17.0	17.0	17.0
	Java	16	16.0	16.0	33.0
	JavaScript	24	24.0	24.0	57.0
	Other	25	25.0	25.0	82.0
	Python	18	18.0	18.0	100.0
	Total	100	100.0	100.0	

5.1.2 Descriptive Statistics

The descriptive statistics for the key variables —code suggestion quality, task complexity, tool usability, user expertise, and developer productivity as shown in Table 10—indicate that participants held positive perceptions exceeding expectations. All five variables had means above 4.03 on a 5-point Likert scale. Tool usability received the highest mean rating of 4.054, suggesting respondents found the AI tools easy to use, seamlessly integrated into their workflows, and responsive. These were closely followed by code suggestion quality (4.050), user expertise (4.040), and developer productivity (4.04), indicating that developers generally agreed the tools improved their work and matched their skill levels.

Table 10. Descriptive Statistics for Main Study

Descriptive Statistics					
	N	Mini-mum	Maxi-mum	Mean	Std. Devia-tion
Code Suggestion Quality Mean	100	4.0	5.0	4.050	.1936
Task Complexity Mean	100	4.0	5.0	4.034	.1365
Tool Usability Mean	100	4.0	5.0	4.054	.1946
User Expertise Level Mean	100	4.0	5.0	4.040	.1729
Developer Productivity Mean	100	4	5	4.04	.197
Valid N (listwise)	100				

Task complexity had a mean score of 4.034, suggesting that respondents perceived the tools as valuable during more rigorous engagements with APIs or advanced logical reasoning. The low standard deviations for all variables (between .13 and .19) indicate that all learners responded homogeneously, suggesting that there was a shared positive experience with the AI-assisted coding tools across the sample.

5.1.3 Reliability

The internal consistency of each construct was evaluated using Cronbach's Alpha, and all values exceeded the recommended 0.70 benchmarks which confirms the reliability of the questionnaire instrument. In Table 11, productivity showed an exceptionally high reliability score of 0.990, reflecting the cohesion between the five indicators that measured perceived improvements in speed, efficiency, satisfaction, and task output.

Table 11. Reliability Coefficients (Cronbach's Alpha)

Variables	Cronbach's Alpha	Cronbach's Alpha Based on Standardized Items	N of items
Code Suggestion Quality	.930	.931	5
Task Complexity	.806	.813	5
Tool Usability	.911	.910	5
User Expertise Level	.927	.927	5
Developer Productivity	.990	.991	5

Code suggestion quality and tool usability also showed high reliability with Cronbach's Alpha scores of 0.930 and 0.911 respectively. This indicates that respondents evaluating AI tools as to the relevance and appropriateness of the responses, responsiveness of the interface, and setup options assessed these elements with clarity and consensus.

User expertise returned a Cronbach's Alpha of 0.927, suggesting participants self-assessed their proficiency, confidence in decisions made, and overall experience with the AI tools in a reliable manner. Task complexity, albeit having a slightly lower alpha of 0.806, still meets the benchmark for acceptable reliability in capturing the coherence among participants regarding what is understood as complex in coding tasks.All awarded reliability scores affirm the validity of the questionnaire's design while confirming its capability to undergo further statistical exploration, including correlations and regression analyses.

5.1.4 Correlations Analysis

As shown in Table 12 below, this study's correlation analysis indicates that there are multiple strong and significant within-case correlations between some independent factors and the dependent variable, which in this case is developer productivity. Out of all the correlations, the strongest one was observed between the quality of code suggestions and the usability of the corresponding tool ($r = 0.947$, $p < 0.01$). This means that respondents who evaluated code suggestions positively also rated the tools highly in terms of usability. To put it another way, the more that developers are provided with relevant AI suggestions during coding, the greater the balancing integration and usability that they encounter in their development environments.

Table 12. Pearson Correlation Matrix of Variables

		Code Suggestion Quality Mean	Task Complexity Mean	Tool Usability Mean	User Expertise Level Mean	Developer Productivity Mean
Code Suggestion Quality Mean	Pearson Correlation	1	.929**	.947**	.905**	.798**
	Sig. (2-tailed)		.000	.000	.000	.000
	N	100	100	100	100	100
Task Complexity Mean	Pearson Correlation	.929**	1	.919**	.883**	.753**
	Sig. (2-tailed)	.000		.000	.000	.000
	N	100	100	100	100	100
Tool Usability Mean	Pearson Correlation	.947**	.919**	1	.848**	.694**
	Sig. (2-tailed)	.000	.000		.000	.000
	N	100	100	100	100	100
User Expertise Level Mean	Pearson Correlation	.905**	.883**	.848**	1	.747**
	Sig. (2-tailed)	.000	.000	.000		.000
	N	100	100	100	100	100
Developer Productivity Mean	Pearson Correlation	.798**	.753**	.694**	.747**	1
	Sig. (2-tailed)	.000	.000	.000	.000	
	N	100	100	100	100	100
**. Correlation is significant at the 0.01 level (2-tailed).						

A similarly strong correlation was identified in the AI-assisted coding with a developer's expertise level ($r=0.905$, $p < 0.01$). This shows that a more experienced developer is likely to derive greater benefit from AI code suggestions owing to their capability to assess and utilize the tool's output. Coupled with a developer's task complexity ($r=0.929$, $p<0.01$), it becomes evident that more seasoned professionals will find AI's assistance crucial during multi-step logical programming procedures as task complexity rises.

Shifting focus to the dependent variable of developer productivity, the results indicated significant positive relationships with all four independent variables. Specifically, productivity was positively associated with code suggestion quality ($r = 0.798$), task complexity ($r = 0.753$), tool usability ($r = 0.694$), and user expertise ($r = 0.747$). The findings show the strong interaction multifactorial determinants have on perceived productivity changes. Effective design of AI-based code suggestion tools that provide actionable output, alongside the complexity of tasks to be completed and the respective individual developer's skill, directly influence perceived productivity amplifications.

Execution of the correlation analysis confirms the multi-dimensional approach of the study while also validating the study's framework. The strong inter-variable correlations demonstrate that AI-assisted programming is both synergistic and multi-layered; enhancement in one aspect, for instance the precision of code, enhances other aspects like ease of use and overall operational efficiency. The results stress the need to take a wider perspective when assessing the role of AI in software engineering; that productivity is not solely dependent on the adoption of the AI tool, but rather on integration with the demands of the task, expectations of the users, and system intricacies.

5.1.5 Regression

The regression model in Table 13 below demonstrated further refinement in the predictive power of the independent variables concerning developer's productivity. It resulted in an R^2 of 0.683, meaning 68.3% of the variance in productivity is a result of code suggestion quality, task complexity, tool's user friendliness, and user's level of expertise. Moreover, the F-value of 51.195 ($p < 0.001$) in Table 14 shows that the model is significant as a whole, thus confirming the model is useful and interpretable.

Table 13. Regression Model Summary (R, R^2, Adjusted R^2)

Model Summary									
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.826[a]	.683	.670	.126	.683	51.195	4	95	.000
a. Predictors: (Constant), User Expertise Level Mean, Tool Usability Mean, Task Complexity Mean, Code Suggestion Quality Mean									

Table 14. ANOVA Table for Regression Significance

ANOVA[a]						
Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	3.245	4	.811	51.195	.000[b]
	Residual	1.505	95	.016		
	Total	4.750	99			
a. Dependent Variable: Developer Productivity Mean						
b. Predictors: (Constant), User Expertise Level Mean, Tool Usability Mean, Task Complexity Mean, Code Suggestion Quality Mean						

Examining the regression coefficients in Table 15, the strongest driver of productivity was code suggestion quality with a β of 1.162 and $p < 0.001$. This supports earlier works (Tan et al, 2024; Pandey et al, 2024) arguing that developers tend to achieve significantly better productivity when they understand and trust the AI-generated code.

In an unexpected turn, tool usability had a negative coefficient of -0.776 ($p = 0.001$) which, when controlling for other factors, suggests that higher tool usability does not always lead to higher productivity. This finding suggests that users who regard tools as simple to operate may become

too dependent on them and therefore disengage from deep analysis, debugging, or critical thinking processes. Or, it could simply point to multicollinearity among the predictors.

Table 15. Regression Coefficients (Standardized & Unstandardized)

Coefficients[a]						
Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-.099	.444		-.224	.823
	Code Suggestion Quality Mean	1.315	.261	1.162	5.036	.000
	Task Complexity Mean	.445	.279	.277	1.593	.115
	Tool Usability Mean	-.776	.217	-.690	-3.577	.001
	User Expertise Level Mean	.044	.182	.035	.241	.810
a. Dependent Variable: Developer Productivity Mean						

The effect of AI on productivity in the workplace showed a positive but not statistically significant result (0.445, p = 0.115). This indicates that while AI tools are helpful for complex tasks, complexity alone does not drive productivity increases. Finally, expertise showed the lowest beta value of (0.044, p = 0.810), suggesting that the productivity of a given developer does not vary significantly with changes in their experience and with regard to the quality and user-friendliness of AI tools provided.

These results support the hypothesis that the perceived quality of AI code suggestions is the primary determinant of improved productivity, overshadowing even the importance of usability and experience as more peripheral predictive factors.

5.2 Presentation of Findings and Analysis

5.2.1 Overview of Interview Participants

The investigation carried out ten semi-structured interviews with developers from varying levels of experience and different geographical locations. The participants encompassed junior, mid-level, and senior developers who utilized GitHub Copilot, Tabnine and Codeium. The participants possessed programming experience between 1 to 15 years which allowed the comparison of diverse views on the use of AI in software development.

5.2.2 Theme 1: Productivity Gains with AI Code Assistants

5.2.2.1 Reduction in Task Completion Time

Many participants highlighted the significant time savings achieved through AI assistants, especially during repetitive tasks or syntax-heavy operations. Developers mentioned that AI suggestions helped maintain coding momentum without constant reference switching.

“Copilot helps me skip over the small stuff—things like for-loops or JSON parsing get generated instantly, and that saves me hours each week.” (Participant 03)

“Even if I don’t accept the suggestion directly, it gives me direction. I complete features faster because of that.” (Participant 07)

These observations echo the quantitative finding that code suggestion quality had the strongest correlation with productivity (r = 0.798), affirming the real-time impact AI tools have on reducing development delays.

5.2.2.2 Impact on Code Accuracy and Error Reduction

Participants also commented on improved accuracy, especially in well-documented domains like web APIs or authentication.

“It catches little mistakes—missing colons, syntax quirks—that I might miss after hours of coding.” (Participant 06)

“The AI's suggestions often match my intent exactly, especially for boilerplate functions, which reduces my overall debugging workload.” (Participant 01)

However, some noted occasional inaccuracies:

“Sometimes the AI fills in things too confidently—like suggesting a function that doesn’t exist in the imported library.” (Participant 09)

This underscores the importance of human validation, even in high-confidence AI-generated code.

5.2.3 Theme 2: Usability and Integration Experience

5.2.3.1 Seamless Workflow Compatibility

A majority of participants praised the ease of integrating AI tools into their existing development environments.

“It fits right into VS Code; I didn’t even need to configure anything complicated. Suggestions just appear where I need them.” (Participant 04)

“Tabnine is fast and understands what I’m doing in the project—it doesn’t need constant input like a chatbot.” (Participant 10)

This aligns with high quantitative ratings for tool usability (mean = 4.054), affirming that technical friction is minimal for most users.

5.2.3.2 Challenges in Tool Responsiveness or Relevance

Despite overall satisfaction, developers pointed out that responsiveness and context awareness could vary, especially in larger codebases.

“Codeium lags on big files—it’s not great when I need suggestions for functions buried inside complex modules.” (Participant 08)

“Sometimes I type a comment and the suggestion is completely off. Like, I asked for a sorting function and it gave me a database call.” (Participant 02)

Such observations reflect the variance in model performance depending on the programming context and tool maturity.

5.2.4 Theme 3: Trust, Risks, and Skill Dependency

5.2.4.1 Concerns About Over-Reliance and Deskilling

A number of participants expressed concern that prolonged use of AI tools could lead to reduced engagement with core problem-solving tasks.

“Sometimes I feel like I’m not thinking through logic as deeply. I just check if the suggestion runs, instead of writing from scratch.” (Participant 05)

“Junior developers in my team rely on Copilot too much—they don’t ask why the code works.” (Participant 03)

These views align with concerns noted in literature (Szolderits, 2025) and reinforce the qualitative importance of balancing assistance with critical thinking.

5.2.4.2 Verification, Ethics, and Responsibility

Participants shared diverse strategies for validating AI-generated code, along with ethical considerations regarding open-source training data and accountability.

“I double-check everything. If something feels too easy, I add extra test cases or compare it with documentation.” (Participant 06)

“I worry about using Copilot in enterprise code. If it generates a snippet from a GPL library, who’s liable?” (Participant 01)

These insights echo the growing call for ethical and secure integration of AI coding assistants in professional software development environments.

6 Discussion

6.1 Interpretation of Key Results

The results of this study firmly validate the hypothesis that AI-enabled code assistants, especially GitHub Copilot and similar applications, substantially increase a software developer’s productivity. A combination of correlation and regression analysis revealed that a developer’s productivity is determined not only by the sophistication of the AI tool but also by its integration with the developer’s workflow, the task’s complexity, and the developer’s skill level.

Perhaps the most important findings came from the correlation analysis, which determined the proposition most strongly correlated to developer productivity was the code suggestion quality ($r = 0.798$). This result confirms that developers appreciate contextually accurate, syntactically correct, and logically coherent propositions provided by AI. When an AI tool reduces the suggestive gap between what is provided and what the programmer needs to execute, it enhances the developer’s ability to deliver tasks promptly and accurately. This was further confirmed in the regression analysis, which found that only code suggestion quality had a significant impact on productivity ($\beta = 1.162$, $p < 0.001$) and explained a considerable amount of the variance in developer output.

Furthermore, the study showed that productivity is strongly influenced by tool usability, user expertise, and task difficulty (with r values between 0.694 and 0.753) though their individual impacts differ within a regression model. While usability and productivity are positively correlated, usabil-

ity shows a negative regression coefficient in the model β = -0.690, p = 0.001. This paradoxical outcome might imply that ease of use fosters a mindset where one assumes the work requires little effort to engage with substantively, resulting in inadequate interaction with the code developed by AI—instead of rigorous, critical appraisal assuming the output is perfect. It may also be a consequence of a suppressor effect from collinearity with other independent variables.

User expertise's lack of measurable impact in the regression model (β = 0.035, p = 0.810) stands out as well. While more seasoned developers tended to be more confident and skilled with evaluating AI-generated code (as the correlation analysis indicated), their skill did not predict greater productivity once other factors were held constant. This may indicate that AI-powered tools create a baseline competence across experience levels, delivering significant value to all developers, though in different forms: novices receive scaffolding; experts see a reduction in the time required for tasks previously burdened by complexity.

In general, the AI tools are found to heighten productivity according to the ecosystem framework of productivity enhancement outlined in the study, thus explaining why the analysis aligns with the productivity hypothesis. Furthermore, these findings support the hypothesis that productivity benefits are not a consequence of only adopting the tools but rather an ecosystem interplay of the interactions with tool caliber, user actions, task transaction difficulty, and overall system usability.

6.2 Pros and Cons of AI Assistants

The advantages of AI-powered coding assistants are numerous, many of which this study has verified empirically. Of these benefits, perhaps the most impactful is acceleration of development processes. Developers reported greater throughput and reduced time spent on routine, manual, or repetitive tasks such as template coding, syntax searching, and routine cycles of coding implementation. These improvements were most pronounced in cases of repetitive work or set design paradigms where AI-powered assistants could serendipitously create appropriate code snippets through automation requiring minimal user input.

Improved AI assistance with learning and onboarding pertains specifically to junior developers and students. The AI-suggested-based guidance helps capture the attention of less experienced developers and provides feedback in real-time coding, thus reducing their reliance on documentation,

mentorship, or manual documentation and senior guidance. Participants also pointed out that AI tools lowered the resource cost in terms of time and effort required, which encouraged the iterative design processes like experimentation and prototyping.

Usability of the tools was rated high as well with a mean of 4.05 implying that the majority of users considered AI assistants to be easy to use and seamlessly fitted into their development environments. The tools' integration with major IDEs, prompt syntax-based responses, and learning through coding context offered a smooth user experience. This corresponds to earlier Pinto et al. (2024) and Tan et al. (2024), who emphasized usability as one of the most important factors driving AI adoption.

On the other hand, the study highlighted some limitations and issues that pertain to the use of AI assistants. Primarily, there is growing concern with the suggestion of AI, especially AI-generated suggestions, and the likely risk of coming to depend on such tools too heavily, especially for junior developers. While the tools may produce valid code, which is syntactically correct, it does not follow that the output is guaranteed to be best-practice compliant, optimized, or free from logical slippages. This tends to produce a dangerous reliance and complacency which undermines the thinking and problem-solving that developers do, especially if they begin to accept suggestions without the critical review.

Another critical concern is the emergence of insecure or plagiarized code. Vulnerable code and copyrighted materials snippets can be traced to open-source repositories, which poses ethical and legal implications for developers leveraging AI-generated code in private or commercial applications. Klemmer et al. (2024) and Lai (2024) have documented such risks and propose increased accountability and validation mechanisms for AI-generated code.

Regarding user experience, some developers reported exasperation with tools suggesting unhelpful stock answers or irrelevant custom responses tailored to more complex problem-solving challenges. Tools performed exceedingly well on rote tasks but showed limited assistance on tasks involving specialized knowledge, intricate systematic reasoning, multi-step logic architectures, or subtle domain nuances. This AI limitation indicates that while these systems can assist human developers, they cannot yet reenact experienced judgment and team design collaboration.

6.3 Future Trends in AI for Development

The integration of AI technologies comes with additional avenues of exploration in the software development field. As developer-AI interaction evolves, there are several factors that Robert et al. (2023) predict will facilitate this change. In particular, the authors mention that we should expect a shift toward more adaptive and personalized AI. In the future, coding assistants can learn the habits of individual developers from their previous projects and suggest relevant and contextual recommendations based on the task at hand. In addition to adaptive and personalized AIs, the authors also noted that productivity would be further enhanced by the implementation of multimodal AIs. For instance, AI that can offer verbal instruction, process written commands, construct visual models, and speak would be far more powerful than contemporary AI. According to Robert et al. (2023), developers could give verbal or diagrammatic instructions and receive syntactically valid, executable code.

The most profound advancements could come from the formulation of universally accepted principles AIs would follow, with audit-proof mechanisms in place alongside unbreachable security routines. Such tools would work in the interests of transparency, reproducibility, and programmers working with AI-enabled components in safety-critical systems. Programs such as Codeium excel in the promotion of privacy-centric, transparent, and ethical use of open-source filters, paving the road for further advancements in socially-responsible AI.

The development of team-aware AI is also expected. AI assistants today function mostly as solo tools for practitioners. Advanced systems can be developed that leverage concepts of team conventions, shared repositories, and real-time collaboration tools to enable them to generate code in a way that meets team goals and improves multi-user collaborations. The use of AI technology in teaching and training developers is expected to grow as well. Schools may start adopting AI coding assistants into class syllabi not only to teach computer programming but also to promote its responsible use, encouraging thorough AI and code review based on sound principles. This change could transform instructional design as future developers will learn coding not as passive adherents to rules of syntax but as active partners in the development of intelligent systems.

7 Conclusion and Recommendations

7.1 Summary of Findings

This study aimed to analyze the impact an AI tool has on improving developer productivity from multiple perspectives while focusing on GitHub Copilot. To meet the objectives, a survey was conducted with 100 developers, the productivity outcomes were measured in accordance to factor evaluation on course of suggestion assessment, ease of use, competence, and level of difficulty alongside an SPSS statistical analysis on the gathered data. The participants' responses affirm the previously established theory that code suggestive feedback directly compared to the developer's action heavily influences his productivity. Accurate suggestions along with context-based reasoning and logic are some of the factors where the developer places great value when utilizing a tool. Other factors such as usability, user experience, and task complexity albeit in a controlled environment matter contribute but much less. High internal consistency across all variables (Cronbach's Alpha > 0.80) emphasizes the precision of the instrument employed whereas the interdependent correlation reveals the reliance among constructs. The primary focal point of this study serves as an insight into the impact of AI technologies and tools which aid in enhancing productivity for all levels of professionals. Irrespective of the experience level, all users are prone to benefits, though the magnitude differs. Definite guidance and assurance are outcomes for novices, while speed and less effort are noticed by experts. The other side of the coin displays the issues of minimal dependence, security, and the sustained long-term maintainability which indicate the need for firm organizational policies that govern its use.

7.2 Best Practices for Using AI Coding Tools

To use AI coding tools efficiently, developers should adopt certain practices. Firstly, it is important to evaluate the quality of generated code thoroughly. Acceptance of code produced by AI without scrutiny is not permitted. Offering executable suggestions requires reasoning AI processes at each step, thus logic-based reasoning and security checks are a must for every AI-generated piece of code output.

Keeping Up with Industry Changes is also essential. Developers—more so those that are yet to advance in their careers—should be equipped with sufficient understanding that enables critical reflection upon suggestions offered by AI in order to provide adequate learning scaffolds.

Furthermore, AI tools should be Integrated with Secure Development Lifecycle. It is imperative that Development Organizations incorporate protocol for static code examinations and versioning systems with AI tools defined by your organization sharpened to meet rigorous standards of comprehensiveness and validation.

Choosing appropriate tool based to the type of the task is another crucial step. Each tool integrates its own set of benefits, and therefore a singular solution would fail achievement of targeted goals. Prioritization of Task Automation offered by GitHub Copilot due to its real-time response capabilities could be favored over Tools like Codeium which may prove invaluable in settings that require attention to privacy.

Last but not least, Ethical standards should be maintained. AI-powered software coding requires great attention to compliance boundaries; the person deploying the model has prior awareness concerning the risks of the software copyright licenses and the model training data sources.

7.3 Final Reflection

The results of this investigation represent a landmark innovation in tracing the relationship shift between software developers and artificial intelligence. AI-powered code assistants have come of age; they are increasingly becoming essential features of contemporary development ecosystems. The benefits of bolstering, speeding up, and enriching the development endeavors are undisputed, but the outcome is not purely beneficial and does come with some caveats. This research emphasizes the need for AI to be intentionally integrated as a collaborative partner to human developers, rather than perceived as a substitute. It is critical for the development community to prioritize productivity alongside ethical responsibility, the preservation of knowledge, and human creativity as the industry transitions towards greater automation. Respecting these aspects will enable software engineering to be revolutionized by AI tools that truly enhance human capabilities—in this new paradigm, machines will generate code, while humans will exclusively devise the logic structures, innovations, and moral frameworks.

References

Addagalla, S. R. D. (2025). *Engineering in the age of AI: Leveraging Copilot for enhanced software development*. *International Journal of Engineering and Advanced Technology Studies, 13*(1), 29–43.

Ajiga, D., Okeleke, P. A., Folorunsho, S. O., & Ezeigweneme, C. (2024). *Enhancing software development practices with AI insights in high-tech companies* (Technical Report TR-2024-003). IEEE Software Engineering Institute.

Chen, A., Huo, T., Nam, Y., Port, D., & Peruma, A. (2024). *The impact of generative AI-powered code generation tools on software engineer hiring: Recruiters' experiences, perceptions, and strategies*. *arXiv preprint arXiv:2409.00875*. https://doi.org/10.48550/arXiv.2409.00875

Cheng, R., Wang, R., Zimmermann, T., & Ford, D. (2024). "It would work for me too": How online communities shape software developers' trust in AI-powered code generation tools. *ACM Transactions on Interactive Intelligent Systems, 14*(2), 1–39. https://doi.org/10.1145/3644815.3644949

Corso, V., Mariani, L., Micucci, D., & Riganelli, O. (2024, April). Generating Java methods: An empirical assessment of four AI-based code assistants. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension* (pp. 13–23). https://doi.org/10.1145/3644815.3644949

Dohmke, T., Iansiti, M., & Richards, G. (2023). *Sea change in software development: Economic and productivity analysis of the AI-powered developer lifecycle*. *arXiv preprint arXiv:2306.15033*. https://doi.org/10.48550/arXiv.2306.15033

Efthimia, M., Eleni, V., Theofanis, K., Venetis, K., & Papakostas, G. A. (2025). AI-powered software development: A systematic review of recommender systems for programmers. *Computers, 14*(4), 119. https://doi.org/10.3390/computers14040119

Glushkova, D. (2023). *The influence of artificial intelligence on productivity in software development* (Doctoral dissertation, Politecnico di Torino). https://webthesis.biblio.polito.it/28435/

Hajam, A. A., Sania, U., & Pradaeva, O. (2024). *AI tools for tracking and enhancing productivity: Next-gen performance management*.

Hamza, M., Iqbal, M. W., & Ahmad, S. Z. (2024). AI-driven assistants' potential for scaled agile software development. *Bulletin of Business and Economics, 13*(2), 974–982.

Jayaraman, K. D., & Singh, P. (2024). *AI-powered solutions for enhancing .NET Core application performance*.

Kalava, S. P. (2024). Enhancing software development with AI-driven code reviews. *North American Journal of Engineering Research, 5*(2).

Klemmer, J. H., Horstmann, S. A., Patnaik, N., Ludden, C., Burton Jr, C., Powers, C., Massacci, F., Rahman, A., Votipka, D., Lipford, H. R., & Rashid, A. (2024, December). Using AI assistants in software development: A qualitative study on security practices and concerns. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security* (pp. 2726–2740). https://doi.org/10.1145/3644815.3644949

Koç, O., Yücedağ, İ., & Şentürk, Ü. (2025). The impact of artificial intelligence enhanced no-code software development platforms on software processes: A literature review. *Duzce University Journal of Science and Technology, 13*(1), 383–401.

Kolusu, S. R. (2024). AI-powered solutions in computer science: A comprehensive COPRAS evaluation. *Intelligence, 3*, 1. https://doi.org/10.46632/jdaai/3/1/9

Kuhail, M. A., Mathew, S. S., Khalil, A., Berengueres, J., & Shah, S. J. H. (2024). “Will I be replaced?” Assessing ChatGPT's effect on software development and programmer perceptions of AI tools. *Science of Computer Programming, 235*, 103111. https://doi.org/10.1016/j.scico.2024.103111

Lai, U. (2024). *ChatGPT’s code suggestion accuracy evaluation*. [Preprint or internal report].

Liang, J. T., Yang, C., & Myers, B. A. (2024, February). A large-scale survey on the usability of AI programming assistants: Successes and challenges. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering* (pp. 1–13). https://doi.org/10.1145/3597503.3608128

Ng, K. K., Fauzi, L., Leow, L., & Ng, J. (2024). *Harnessing the potential of Gen-AI coding assistants in public sector software development*. *arXiv preprint arXiv:2409.17434*. https://doi.org/10.48550/arXiv.2409.17434

Pandey, R., Singh, P., Wei, R., & Shankar, S. (2024). Transforming software development: Evaluating the efficiency and challenges of GitHub Copilot in real-world projects. *arXiv preprint arXiv:2406.17910*. https://doi.org/10.48550/arXiv.2406.17910

Piltan, N. (2024). *Enhancing efficiency in test-driven development through artificial intelligence* (Master's thesis). https://www.duo.uio.no/bitstream/handle/10852/112595/5/Master-thesis---Nima.pdf

Pinto, G., De Souza, C., Rocha, T., Steinmacher, I., Souza, A., & Monteiro, E. (2024, April). Developer experiences with a contextualized AI coding assistant: Usability, expectations, and outcomes. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering—Software Engineering for AI* (pp. 81–91). https://doi.org/10.1145/3644815.3644949

Qianyi, Y. (2024). *Systematic evaluation of AI-generated Python code: A comparative study across progressive programming tasks*. https://doi.org/10.21203/rs.3.rs-4955982/v1

Robert, H., Langlois, S., & Bentley, T. (2023). *Emerging trends in AI-assisted software development*. *Journal of Emerging Software Practices, 12*(3), 101–117.

Sager, J., & Wahab, A. (2024). *Enhancing software development with AI: Measuring productivity and automation impact*. https://www.researchgate.net/publication/388834928

Saklamaeva, V., & Pavlič, L. (2023). The potential of AI-driven assistants in scaled agile software development. *Applied Sciences, 14*(1), 319. https://doi.org/10.3390/app14010319

Shah, K., & Trehan, A. (2024). *Streamlining software development: A comparative study of AI-driven automation tools in modern tech*. [Conference paper or unpublished manuscript].

Sherje, N. (2024). Enhancing software development efficiency through AI-powered code generation. *Research Journal of Computer Systems and Engineering, 5*(1), 1–12. https://www.technicaljournals.org/RJCSE/index.php/journal/article/view/90

Solanke, A. A. (2023). *Generative AI's impact on enterprise software development lifecycles: A framework for integration, governance and ROI measurement*.

Szolderits, C. M. (2025). *A new era of coding: Investigating GitHub Copilot's influence on productivity, code quality and ethics in software development* (Doctoral dissertation, Alpen-Adria-Universität Klagenfurt).

Tan, X., Long, X., Ni, X., Zhu, Y., Jiang, J., & Zhang, L. (2024). How far are AI-powered programming assistants from meeting developers' needs? *arXiv preprint arXiv:2404.12000*. https://doi.org/10.48550/arXiv.2404.12000

Upadhyaya, N. (2024). *AI-powered secure software development: The future of safe and efficient coding*.

Veeramachaneni, V. (2021). AI-driven software development: Enhancing code quality and maintainability through automated refactoring. *International Journal of Recent Developments in Science and Technology, 5*(6), 94–101.

Appendices

Appendix 1. Quantitative questionnaire

Section A: Demographic Information

Please select or fill in the most appropriate response for each question:

1. **Age:**

   ☐ Under 20 ☐ 21–30 ☐ 31–40 ☐ 41–50 ☐ Over 50

2. **Gender:**

   ☐ Male ☐ Female

3. **Country of Residence:**

4. **Years of Programming Experience:**

   ☐ Less than 1 year ☐ 1–2 years ☐ 3–5 years ☐ 6–10 years ☐ More than 10 years

5. **Current Role:**

   ☐ Student ☐ Junior Developer ☐ Mid-Level Developer ☐ Senior Developer ☐ Team Lead / Architect

6. **Have you previously used AI coding assistants (e.g., GitHub Copilot, Tabnine, Codeium)?**

   ☐ Yes ☐ No

7. **Which AI coding assistant do you primarily use?**

   ☐ GitHub Copilot ☐ Tabnine ☐ Codeium ☐ Other (please specify): _____

8. **Primary Programming Languages Used:**

☐ JavaScript ☐ Python ☐ Java ☐ C# ☐ Other: _____

Section B: Main Questionnaire (Likert Scale: 1 = Strongly Disagree to 5 = Strongly Agree)

1. Developer Productivity

Source: Sherje (2024); Dohmke et al. (2023)

9. Using an AI code assistant helps me complete development tasks more quickly.

10. I produce fewer coding errors when using an AI assistant.

11. My overall development efficiency has improved with AI assistance.

12. I can complete more tasks per session when using AI-generated suggestions.

13. I feel more satisfied with my productivity when using an AI assistant.

2. Code Suggestion Quality

Source: Tan et al. (2024); Szolderits (2025)

14. The code suggestions are relevant to the context of my task.

15. The syntax of AI-suggested code is usually correct.

16. AI-generated functions meet my intended logic or requirements.

17. I rarely have to correct or rewrite the suggested code.

18. The AI suggestions match best practices or common usage patterns.

3. Task Complexity

Source: Shah & Trehan (2024); Kalava (2024)

19. The AI assistant performs well even when tasks are complex.

20. I rely more on AI tools for tasks involving unfamiliar libraries.

21. The tool is helpful when dealing with new or unknown programming frameworks.

22. I find AI assistance more useful in back-end tasks than front-end tasks.

23. I tend to use AI more when working on time-consuming or repetitive tasks.

4. Tool Usability

Source: Pinto et al. (2024); Ng et al. (2024)

24. The AI tool integrates well with my development environment (e.g., VS Code).

25. The tool interface is easy to navigate and interact with.

26. I can easily enable, disable, or adjust settings in the AI assistant.

27. The tool responds quickly and does not slow down my workflow.

28. The AI suggestions are easy to insert or modify in real time.

5. User Expertise Level (Self-Assessment)

Source: Liang et al. (2024); Cheng et al. (2024)

29. I am confident in evaluating the correctness of AI-generated code.

30. I understand when to accept or reject AI suggestions.

31. I have experience using at least two different AI code assistants.

32. I can solve most programming problems without relying heavily on AI.

33. I actively review and test AI-generated code before using it.

Appendix 2: Semi-Structured Interview Sheet

Research Title:

Evaluating the Effectiveness of AI-Powered Code Assistants (e.g., GitHub Copilot) in Improving Developer Productivity

Interview Type:

Semi-Structured (Estimated Duration: 30–40 minutes)

Participant Information:

Role: _____

Experience Level: _____ years

Preferred AI Tool: _____

Section A: Developer Productivity and Efficiency

1. Can you describe how using an AI-powered code assistant has impacted the speed at which you complete coding tasks?
2. In what ways, if any, has the use of AI assistants helped reduce repetitive or time-consuming aspects of your development process?
3. Have you noticed any changes in your error rates, code quality, or time spent on debugging since using AI tools?
4. To what extent do you feel these tools have increased or decreased your overall productivity?

Section B: Tool Comparison and Usage Experience

1. Which AI-powered coding tool(s) have you used, and how do you compare their accuracy and performance?
2. How well do these tools integrate into your existing development workflow or IDE?
3. Can you share an example of a task where the AI assistant performed exceptionally well—or poorly?
4. What factors influence your choice of which tool to use for specific types of development tasks (e.g., full-stack, front-end, back-end)?

Section C: Limitations, Risks, and Perceptions

1. Have you encountered any limitations, risks, or reliability issues while using AI-generated code?

2. How do you typically verify or trust the code suggestions provided by these assistants?

3. In your opinion, how do these tools affect your own coding skills, creativity, or problem-solving ability?

4. What improvements or safeguards would you recommend to make these AI assistants more effective or responsible to use?