JOURNAL OF COMPUTATIONAL INNOVATION AND ANALYTICS
https://e-journal.uum.edu.my/index.php/jcia

How to cite this article:
Aziz, N. K. (2026). Technology acceptance of GitHub Copilot: A study among computing students. *Journal of Computational Innovation and Analytics*, *5*(2), 117-133. https://doi.org/10.32890/jcia2026.5.2.7

# Technology acceptance of GitHub Copilot: A study among computing students

**Nian Khidr Aziz**
Department of Electrical Engineering, College of Engineering
Salahaddin University-Erbil, Kurdistan Region, Iraq

*Corresponding Author: nian.aziz@su.edu.krd*

Received: 23/05/2026 Revised: 14/07/2026 Accepted: 16/07/2026 Published: 31/07/2026

## ABSTRACT

AI pair programmers (AIPPs) are becoming increasingly used in software development environments. This is encouraging universities to examine the role of these tools in computing education and programming courses to ensure that their programs remain market-oriented. However, despite the rapid adoption of AI pair programmers, limited empirical evidence exists regarding students' acceptance and educational use of these tools in authentic project-based software development environments. One of the most widely used AIPPs is GitHub due to its integration with many development environments. This study investigates the Technology Acceptance (TA) of GitHub Copilot among computing students. We conducted a mixed-methods study with 114 senior computer science and software engineering students from three public universities in the Kurdistan Region of Iraq. The task was to develop a static-responsive portfolio website using GitHub Copilot as an AI pair programmer. The Technology Acceptance Model (TAM) was used to evaluate Perceived Ease of Use (PEU), Time Efficiency (TE), Support in Learning (SiL), Code Quality (CQ), and Willingness to Recommend (WtR). Quantitative descriptive statistics and thematic analysis of qualitative responses were used to evaluate these features. Findings indicate high positive initial acceptance. Thematic analysis showed that students value GitHub Copilot for overcoming syntax barriers and reducing development time, but also express concerns about understanding and over-reliance. This study presents one of the first empirical mixed-methods examinations of AIPP acceptance in computing education. The contribution of the study lies in identifying the dual nature of this acceptance: high enthusiasm tempered by reflective concerns about deep learning. The results suggest that Copilot can improve efficiency in programming courses, but its use should be balanced to avoid superficial dependency on the tool.

**Keywords**: AI pair programmer, Copilot, GitHub, Perceived Usefulness (PU), Technology Acceptance Model (TAM)

## INTRODUCTION

The emergence of AI models trained on codebases has added the power to write, suggest, and evaluate code, changing the way programmers write and manage programs (Zhao et al., 2026). AI-assisted coding tools referred to as AI-Pair Programmers (AIPP) are based on Large Language Models (LLMs) trained on large code corpora rather than only natural language corpora (Sergeyuk et al., 2025). In some cases, large codebases are added to their language resources. These models generate code in different programming languages based on the programmer's prompt (Bañuls-Lapuerta et al., 2026). However, the quality, functionality, and optimality of the code produced by these tools are still under discussion. These tools are gaining rapid popularity in the software development industry and revolutionizing how developers interact with code (Revuri et al., 2025).

While AI-pair programmers provide significant productivity gains for professional developers, they present a challenge for computer education. Computer schools prepare students for a workforce where AI-pair programmers are ubiquitous, but foundational programming skills and critical problem-solving are not eroded by over-reliance on automation(Andersen et al., 2026). Current research on AI-assisted coding has mainly focused on the correctness and efficiency of isolated tasks. There is a significant gap in how students experience these tools within authentic project-based learning (PBL) practice (Xu et al., 2024). There is a lack of insight into the interplay between students' perception, their actual performance, and their capacity for critical engagement-the ability to review, debug, and modify the AI-generated code rather than accept it passively(Groothuijsen et al., 2024).

Furthermore, relatively few studies have investigated the use of GitHub Copilot throughout an entire software development project involving computer science and software engineering students. Most existing research has relied on theoretical evaluations, controlled experiments, benchmark comparisons, or short programming exercises, rather than examining students' sustained interaction with AI pair programmers in real-world software development activities(Ghorbian et al., 2026). Hence, little is known about the relationship between students' perceptions, their actual programming performance, and their ability to critically engage, i.e., to review, debug, modify, and validate AI-generated code instead of simply accepting it.

This study addresses this gap through a mixed-method field study. The study involves 114 senior undergraduate students from computer science and software engineering programs. Participants used GitHub Copilot to develop a static-responsive website. After completing the task, students documented their experience, including perceptual survey data, objective performance metrics, and qualitative reflections on their critical engagements. Based on Technology Acceptance Model (TAM)(Saif et al., 2024), this study tries to answer three questions:

1. How do students accept GitHub Copilot in terms of ease of use, perceived code usefulness and learning support, code quality, and time efficiency?
2. What relationship exists between the perceptions and objective measures of performance?
3. What theme characterizes students' critical engagement with AI-generated code?

With the emergence of the Internet, programmers increasingly started using online resources, such as code repositories, blogs, and community platforms, to find code snippets, functions, methods, classes, libraries, packages, and codebases that would help them resolve their software problems. Many support

sites are now established to help programmers (Sîrbu & Czibula, 2025a). Recently, Large Language Models (LLMs) have become a formidable tool for generating programming code. Strategies like large NLP datasets, new algorithms (Sorin et al., 2020), and innovative NLP units (Künas et al., 2023) all support AIPPs' mission.

Based on these considerations, this study investigates the Technology Acceptance of GitHub Copilot among senior students of computer science and software engineering in a mixed-methods study based on the Technology Acceptance Model (TAM). A total of 114 students from three public universities participated in an authentic Project-Based Learning (PBL) activity, where GitHub Copilot was used as an AI pair programmer throughout the development of a complete software project. Quantitative analysis of students' perceptions and objective programming performance was complemented by qualitative thematic analysis of their reflective experiences. The results offer empirical evidence on students' acceptance of GitHub Copilot and practical insights for educators and curriculum designers willing to integrate AI-assisted programming tools into computing education.

## LITERATURE REVIEW

The AI Pair Programmers (AIPPs) have been widely studied in recent years, especially in the training and education sector (Gəruslu et al., 2026). As a new technology, the functionalities, limitations, and acceptance of these tools have not yet been carefully understood (Döderlein et al., 2025). Among many available tools, GitHub Copilot and ChatGPT are the most studied tools (Revuri et al., 2025; Sorin et al., 2020). Moradi Dakhel et al. (2023) studied the performance, code accuracy, and problem-solving capabilities of AIPPs in general, and Kuhail et al. (2024) studied ChatGPT in particular. Most of these studies show that AIPPs provide incomplete and inaccurate solutions for more intricate tasks, concluding with the imperative for human verification. Researchers and practitioners in coding education pointed to technical advances that may be applied to AIPPs (Palaniappan et al., 2025). Sîrbu & Czibula (2025b) presented Abstract Syntax Graphs for improving coding learning; Pan et al. (2025) proposed Expert Encoder Group (EEG) for better code efficiency; Yan et al. (2025) discussed improving redundancy by *RRGcode*; and Gou et al. (2024) presented the COCO approach to maintain consistency. From a programming course viewpoint, ChatGPT helped students perform better in large data processing (Park & Kim, 2025), while the risks of heavy use of AI-generated code in academic environments exist; to locate this sort of code, Nguyen et al. (2024) suggested GPTSniffer.

AIPP affects the software development lifecycle and the role of software engineers in the near future (Almeida et al., 2024). GPT plugins can improve the flagging of logical and grammatical errors, and PyCoder, a syntax-aware code completion tool, overcomes system constraints (Takerngsaksiri et al., 2024). In domains beyond general programming, for example, several LLMs, including ChatGPT, have been shown to create deep learning models for time series analysis. Some software engineering issues, such as sensitivity, prompt engineering, and negative coding propagation, have been studied from AIPP's point of view (Go et al., 2023; Gopali et al., 202). These studies reveal the opportunities and challenges in utilizing AIPP in software development processes.

However, as it is an emerging technology, the existing literature on AIPP provides insights into the functional and technical issues of using AIPP tools, but a focused investigation into computing students' own perceptions and lived experiences within structured, project-based learning remains underexplored. Relatively few studies have investigated the use of GitHub Copilot throughout an entire software development project involving computer science and software engineering students. Most existing research has relied on theoretical evaluations, controlled experiments, benchmark comparisons, or short

programming exercises, rather than examining students' sustained interaction with AI pair programmers in real-world software development activities (Ghorbian et al., 2026). Most research in the literature has studied performance metrics or instructor viewpoints, leaving a gap in understanding how students perceive the AIPP's ease of use, its impact on their learning process, their trust in its code quality, and the critical reasoning it provokes (Zhao et al., 2026).

This study aims to address these gaps by adopting a mixed-methods, perception-oriented approach based on the Technology Acceptance Model (TAM). It combines students' perceptions, objective measures of programming performance, and qualitative reflections by leveraging a publicly available multi-university dataset collected from real-world project-based software development activities. Unlike previous studies that have mainly relied on theoretical evaluations, controlled experiments, or isolated programming exercises, this study investigates students' continued use of GitHub Copilot during the development of an entire software project. Furthermore, it provides one of the first comprehensive empirical assessments of GitHub Copilot among computer science and software engineering students by integrating subjective perceptions, objective performance measures, and critical reflections within a unified evaluation framework.

## METHODOLOGY

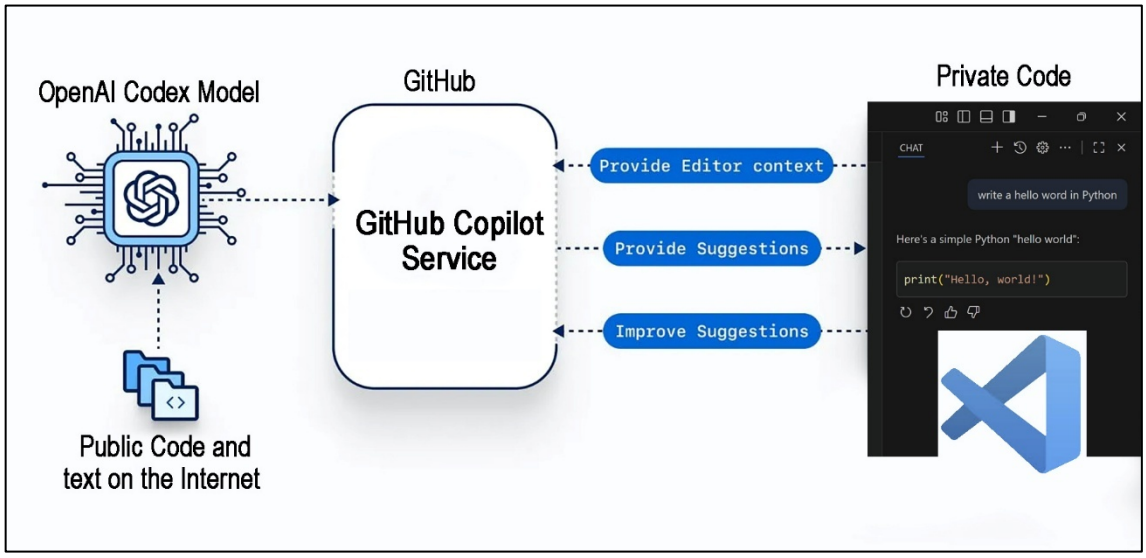
### Study Design and Participants

This research employed a well-known Project-Based Learning (PBL) strategy to conduct a field study. A cohort of 114 senior-level undergraduate students (Male=77, Female=37) from computer science and software engineering programs from three public universities in the Kurdistan Region of Iraq participated in the study. The study organized a development competition to encourage engagement. The task was to develop a static-responsive personal portfolio website to be hosted on GitHub Pages. The study evaluated the students' perception of using the AI-assisted coding tool GitHub Copilot in a complete software project.

Among the AI-assisted tools, GitHub Copilot is a popular and powerful AIPP. It can suggest code snippets, complete functions, and provide documentation support (Oertel et al., 2025). It is supported by the OpenAI Codex language model, a specialized version of the GPT architecture that has been fine-tuned using a massive corpus of publicly available code from GitHub repositories. Copilot is integrated directly into popular development environments like Visual Studio Code, enabling seamless code generation in real time (Moradi Dakhel et al., 2023). GitHub Copilot was first released in technical preview in June 2021 (Husein et al., 2025). It functions by analyzing the context of the current file, user comments, and code patterns to generate code lines, functions, or blocks of code. Figure 1 illustrates GitHub Copilot's working model (Rawat, 2021). The working procedure is as follows: the GitHub Copilot IDE extension sends the programmer's prompt and code to the GitHub Copilot service, which then uses OpenAI Codex to synthesize and suggest code. It reads through all the open-source code on the GitHub repository and tries to find the best possible code related to it (Chen et al., 2021). Among many AIPPs, GitHub Copilot has gained significant attention in both industry and academia, raising questions about its effectiveness, reliability, and the evolving nature of AIPPs in software development (Zhou et al., 2025).

**Figure1**

*The GitHub Copilot's Working Mode*




*Note:* This figure was generated using generative AI based on author-provided prompts and edited by the author for accuracy and clarity.

To ensure consistency of the submissions, participants were provided with a set of structured software requirements. The requirements were formalized as a Software Requirements Specification (SRS) checklist as:

Functional Requirements:

- F1. Content & Structure: Implement Home, About, Portfolio, and Contact sections.
- F2. Assets: Include a downloadable Curriculum Vitae (CV) in PDF format.
- F3. Integration: Provide links to at least two professional social media profiles.

Non-functional Requirements:

- NF1. Responsiveness: Ensure correct rendering on both desktop and mobile viewports.
- NF2. Deployment: Manage code via Git and deploy the final version using GitHub Pages.

For the detailed procedure and the end-to-end workflow from setup to submission, see Algorithm 1.

**Algorithm 1**

*Software Requirements Specification (SRS)*

1. *Setup Phase*

*Setup: Study requirements, initialize Git repository, activate Copilot.*
*Use Copilot interactively for code generation and completion. Commit code iteratively.*

2. *Define project task:*

*Develop a static personal portfolio website*
*Tools: GitHub Copilot and Visual Studio Code*

*3. Specify software requirements:*

*- Downloadable CV*
*- Responsive design for desktop and mobile*
*- Social media integration*
*- Performance optimization*
*- Basic security measures*
*- Deployment using GitHub Pages*

*4. Specify software requirements:*

*- Downloadable CV*
*- Responsive design for desktop and mobile*
*- Social media integration*
*- Performance optimization*
*- Basic security measures*

*5. Project Implementation*

*For each student:*
*a. Use GitHub Copilot as an assistant in coding*
*b. Develop the website according to the provided requirements*
*c. Deploy the final version to GitHub Pages*
*d. Submit the project for evaluation*

*6. Submission: Deploy final website, record completion timestamp.*

*7. Evaluation: Complete post-task survey and qualitative reflection.*

*End*

**Data Collection and Measurements**

After submission of the completed task, participants completed a questionnaire that rated their experience during the development process. Data were collected across the following three streams and summarized in Table 1.

1. Perceptual Survey: A post-task survey using a 5-point Likert scale (Anjaria, 2022) measured five constructs adapted from the Technology Acceptance Model (TAM) represented by rows 1 to 5 in Table 1.

2. Objective Performance Metrics: Actual time to complete the project (in hours) and a project score based on a detailed rubric. See Table 1, referring to rows 6 and 7.
3. Qualitative Feedback: Students were given an open-ended question asking them to describe a critical engagement with an incorrect or suboptimal AI suggestion. Refer to row 8 in Table 1.

The dataset was named GitHub Copilot in Computing Education (GCCE) and is publicly available (Aziz, 2026).

**Table 1**

*Research Constructs, Theoretical Foundations, and Measurement Items*

| ID | Feature | Theoretical foundation | Operational measure / survey item | Data type and scale |
|---|---|---|---|---|
| F1 | Perceived Ease of Use | Technology Acceptance Model (TAM)(Fel et al., 2025) | GitHub Copilot was easy to use. | Ordinal (5-pt. Likert) |
| F2 | Perceived Code Quality | TAM – Perceived Usefulness (Output) | GitHub Copilot helped me write better quality code. | Ordinal (5-pt. Likert) |
| F3 | Perceived Time Efficiency | TAM – Perceived Usefulness (Process) | GitHub Copilot helped me complete the project faster. | Ordinal (5-pt. Likert) |
| F4 | Support in Learning | Educational Scaffolding | GitHub Copilot helped me learn new programming concepts. | Ordinal (5-pt. Likert) |
| F5 | Willingness to Recommend | TAM – Behavioral Intention | I would recommend GitHub Copilot to other students | Ordinal (5-pt. Likert) |
| F6 | Actual Completion Time | Objective Performance Metric | Total project duration (hours) (Submission Timestamp – Start Timestamp). | Ratio (Continuous, in hours) |
| F7 | Final Project Score | Authentic Assessment (Vlachopoulos & Makri, 2024) | Rubric-based assessment across four dimensions. | Score from 0 to 10 points |
| F8 | Critical Engagement | Metacognitive Reflection | Open-ended response: Describe an instance where Copilot's suggestion was incorrect or suboptimal and explain how you addressed it. | Qualitative (Textual) |

## ANALYSIS OF RESULTS AND DISCUSSION

As mentioned before, we structured the collected data (GCCE dataset) in a standard format for analysis. The dataset contains eight features for 114 observations (participants). We used quantitative and qualitative data analysis methods to highlight the findings. Qualitative responses were analyzed using thematic analysis. The responses were independently reviewed, assigned initial codes, grouped into related categories, and subsequently organized into overarching themes through an iterative coding process.

**Student Perceptions of The AI Pair Programmer GitHub Copilot**

Descriptive statistics for the first five constructs, rows 1-5 in Table 1, measured on a 5-point Likert scale (1=Strongly Disagree, 5=Strongly Agree), are detailed in Table 2. Students reported overall very positive perceptions. Perceived Ease of Use scored highest (M = 4.82, SD =0 .38), which means that the tool was highly accessible. Perceived Time Efficiency (M = 4.71, SD = 0.51) and Support in Learning (M = 4.71, SD = 0.46) were also rated very highly. Perceived Code Quality showed strong endorsement with some variability (M = 4.40, SD = 0.66). Willingness to Recommend showed a ceiling effect (M = 5.00, SD = 0.00). All participants selected the maximum score, which reflects positive behavioral intention.

**Table 2**

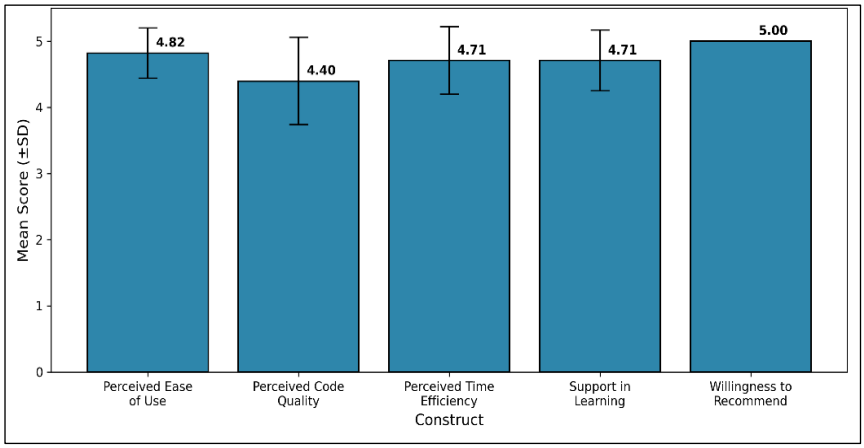
*Summary Statistics of Students' Perceptions*

| Variable | Mean | SD | Interpretation |
|---|---|---|---|
| Perceived Ease of Use | 4.82 | 0.38 | Very high, low variability |
| Perceived Code Quality | 4.40 | 0.66 | High, moderate variability |
| Perceived Time Efficiency | 4.71 | 0.51 | Very high |
| Support in Learning | 4.71 | 0.46 | Very high |
| Willingness to Recommend | 5.00 | 0.00 | Ceiling effect |

Figure 2 shows the visualized overall positive perceptions. The figure shows mean scores with standard deviations for each construct.

**Figure 2**

*Mean Scores and Standard Deviations for Perceptual Constructs (5-point Likert Scale)*




***Objective Performance Outcomes***

Table 3 shows the objective metrics for project completion (rows 6 and 7 in Table 1). The average time for completing the development task was 12.21 hours (SD=1.07). This implies a reasonable variability around the expected time. The mean final project score (based on a detailed rubric, scale 0-10) was 7.86 (SD=0.38), indicating a high level of successful task completion with a moderate variation in quality of output.

**Table 3**

*Summary of Performance Metrics Across Participants*

| Variable | Mean | SD | Interpretation |
|---|---|---|---|
| Actual Completion Time | 12.21 | 1.07 | Reasonably tight distribution |
| Final Project Score (Total) | 7.86 | 0.38 | Moderate-to-high performance |

**Relationships Between Perceptions and Performance**

Spearman's rank-order correlations (Jiang et al., 2024) were calculated to determine the links between the perceptual constructs and the objective performance measures (see Table 4). The Willingness to Recommend option was dropped due to lack of variety. The associations between Perceived Ease of Use and Perceived Time Efficiency ($\rho = 0.28$, $p < .01$) and between Perceived Time Efficiency and Support in Learning ($\rho = 0.36$, $p < .001$) were moderately positive. These values indicate internal consistency in students' attitude reflections.

**Table 4**

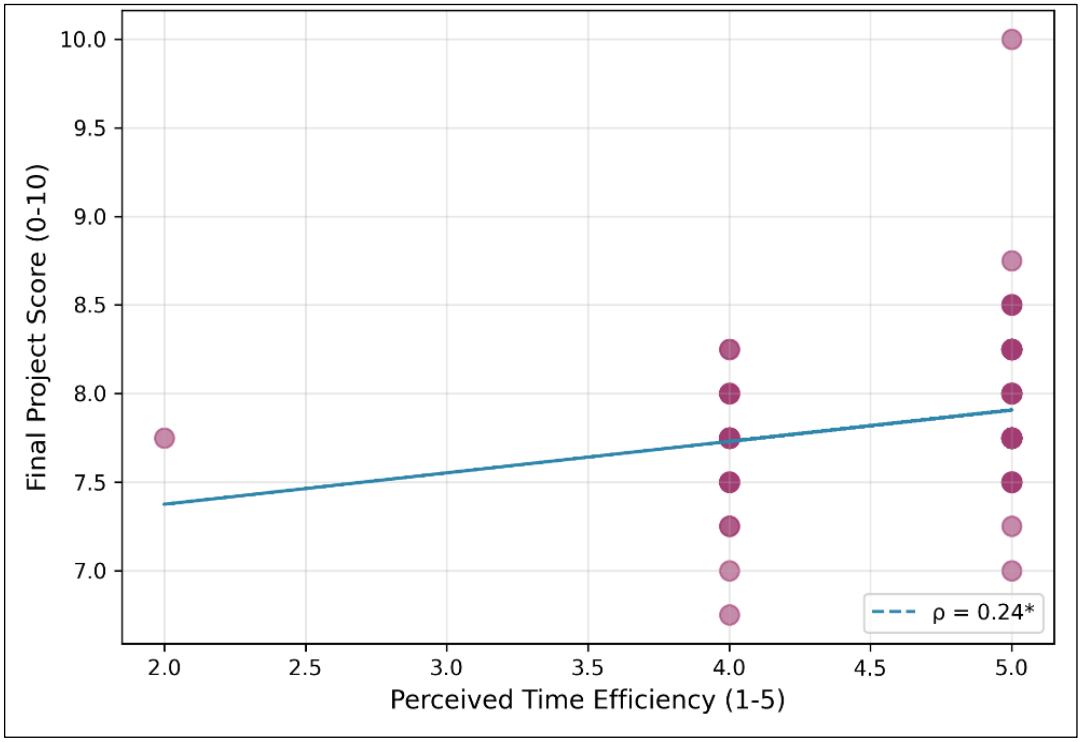
*Spearman Correlations Between Perceptions and Performance Outcomes*

| Variable | Actual completion time (ρ) | Actual completion time (p-value) | Final project score-total (ρ) | Final project score-total (p-value) |
|---|---|---|---|---|
| Perceived Ease of Use | -0.169 | 0.0724 | -0.086 | 0.365 |
| Perceived Code Quality | -0.121 | 0.1986 | -0.127 | 0.177 |
| Perceived Time Efficiency | 0.032 | 0.7385 | 0.24 | 0.010 |
| Support in Learning | -0.037 | 0.6933 | 0.089 | 0.348 |
| Actual Completion time | N.A | N.A | 0.18 | 0.055 |

Figure 4 shows that perceived time efficiency correlates positively with project quality, where higher perceived efficiency corresponds to marginal improvements in project scores.

**Figure 4**

*Scatter Plot of Perceived Time Efficiency Against Final Project Score*




Overall, the associations between perceptions and performance results were limited. Perceived Ease of Use was negatively correlated with task completion time. However, the effect was small ($\rho = -0.169$, $p = .0724$), possibly indicating that students who found the tool easier to use also tended to complete the assignment slightly more quickly. Perceived Time Efficiency has a weak but statistically significant positive relationship with the final project score ($\rho = 0.24$, $p = .01$). Other perception measures were negligibly and non-significantly correlated with measures of performance. Completion time was weakly positively correlated with final project score ($\rho = 0.18$, $p = .055$), showing a marginal trend that students who spent more time on the task tended to produce somewhat higher quality work.

The relationships between perceptions and performance outcomes were generally weak. Perceived Ease of Use showed a small negative association with task completion time ($\rho = -0.169$, $p = .0724$), interpreted as a trend where students who found the tool easier to use may have completed the task slightly faster. Perceived Time Efficiency shows a small but statistically significant positive correlation with the final project score ($\rho = 0.24$, $p = .01$). Other perception measures showed negligible, non-significant relationships with performance metrics. A weak positive correlation was found between completion time and final project score ($\rho = 0.18$, $p = .055$), indicating a marginal trend in which students who spent more time on the task tended to achieve somewhat higher-quality outcomes.

### *Multiple Regression Analysis*

To explore the combined effect of students' perceptions towards programming performance, multiple linear regression analysis was performed, where the final project score was treated as the response variable and Perceived Ease of Use, Perceived Code Quality, Perceived Time Efficiency, Support in Learning, and Actual Completion Time were treated as predictor variables. The results are presented in Table 5, in which the regression model was statistically significant ($F(5,108) = 2.794$, $p = 0.021$), explaining approximately 11.5% of the variance in project performance ($R^2 = 0.115$).

**Table 5**

*Multiple Linear Regression Predicting Final Project Score*

| Variable | Standardized Coeff. (β) | Unstandardized Coeff. (B) | Standard Error (SE) | T-Statistic | p-Value |
|---|---|---|---|---|---|
| Perceived Ease of Use | -0.114 | -0.112 | 0.099 | -1.136 | 0.259 |
| Perceived Code Quality | -0.102 | -0.058 | 0.054 | -1.079 | 0.283 |
| Perceived Time Efficiency | 0.267 | 0.197 | 0.074 | 2.668 | 0.009 |
| Support in Learning | 0.026 | 0.022 | 0.081 | 0.269 | 0.789 |
| Actual Completion time | 0.141 | 0.050 | 0.033 | 1.528 | 0.129 |

Among all predictors, only Perceived Time Efficiency was a significant predictor of higher project scores ($\beta = 0.267$, $p = 0.009$). Students who perceived GitHub Copilot as helping them complete programming tasks more efficiently tended to get higher rubric scores. At the same time, none of the other predictors was statistically significant when controlling for the other variables.

### Thematic Analysis of Students' Critical Engagement

Thematic analysis was carried out through data from open-ended questions about critical engagement. A statistical analysis of the 114 participants' responses yielded five distinct themes; they are summarized in Table 6 and further visualized in Figure 5. The frequency distribution of the themes shows that reflections are dominated by efficiency and quality. The most common theme, Efficiency vs. Quality Trade-Off (f=30), included student reports of Copilot. It suggests functional but suboptimal, verbose, or outdated solutions that they decided to improve. The themes Code Quality & Maintainability Risks (f=16) and Need for Human Oversight (f=11) underline students' roles as critical reviewers. They recognize issues related to code structure, security, and appropriate API use. Less frequent but significant themes included reflections on UX & System Design (f=6) and the tool's potential Impact on Critical Thinking (f=2), where students reported a risk of passive acceptance.

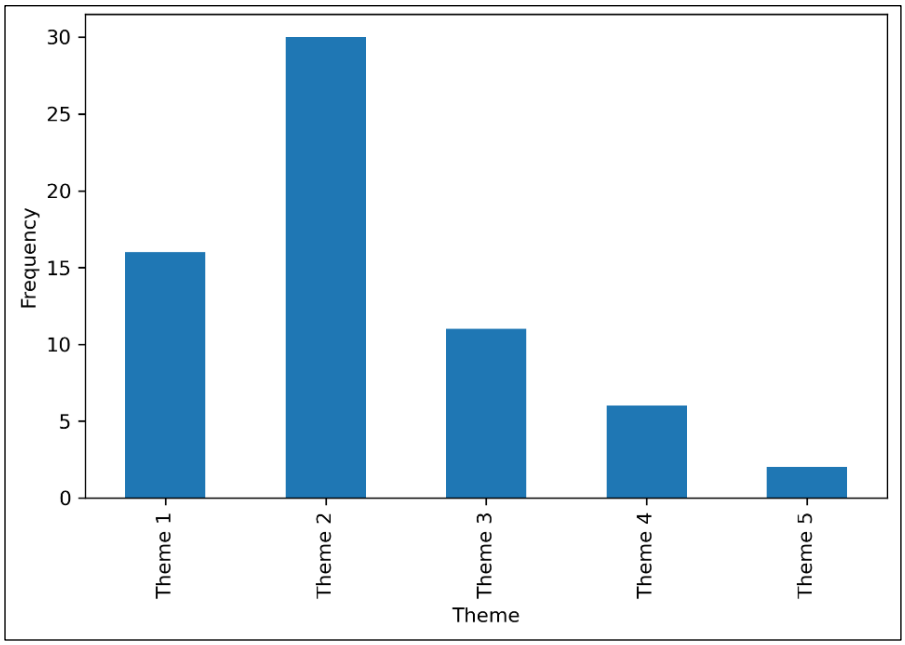
**Table 6**

*Summary of Final Codes and Associated Themes*

| Descriptive code | Theme | Frequency |
|---|---|---|
| Time Efficiency<br>Non-Optimal Solutions<br>Outdated Practices | Theme 1: It is obvious that AIPP affects the software development lifecycle and the role of software engineers in the near future Efficiency vs. Quality Trade-Off | 30 |
| Maintainability & Code Quality<br>Code Complexity | Theme 2: Code Quality and Maintainability Risks | 16 |
| API Misuse<br>Security Concerns | Theme 3: Need for Human Oversight | 11 |
| User Experience (UX) Concerns<br>Performance & Responsiveness | Them 4: UX and System Design Limitations | 6 |
| Critical Thinking Impact | Theme 5: Impact on Critical Thinking | 2 |

**Figure 5**

*Frequency Distribution of Critical Engagement Themes*




**Discussion of Results**

The analysis shows that Willingness to Recommend (WtR) and Perceived Ease of Use (EoU) have high positive perceptions. This provides evidence of adoption of AI pair programmers (AIPP) among computing students according to the Technology Acceptance Model (TAM) (Saif et al., 2024; Tbaishat et al., 2026). The high score of EOU is interpreted as a uniquely low barrier for acceptance in computing education and coding courses. EOU's high score is interpreted as a uniquely low barrier to entry for computing education and coding courses. This is in line with the Technology Acceptance Model

(TAM), which states that technologies that are perceived as easy to use are more likely to be accepted by users (Saif et al., 2024). Besides, it is consistent with the findings of Jazim et al.(2025), who found that perceived ease of use and perceived usefulness have a significant impact on students' intentions to adopt AI-assisted educational technologies. These are key characteristics for software adoption. The high values of Perceived Time Efficiency (TE) and Support in Learning (SiL) are important because they are related to productivity increases (Jazim et al., 2025). These findings reveal that students view Copilot not only as a shortcut tool but also as a multi-purpose tool that accelerates task completion while serving as a dynamic learning support.

On the other hand, the weak relationships between perceptual conceptions and objective performance represent another significant finding. It shows a connection between Perceived Time Efficiency (TE) and higher project ratings, indicating that students who used the tool were more productive. The absence of robust predictive relationships suggests that the tool's help does not directly relate to measurable outcomes in a controlled task. Our thematic analysis makes this gap clear. Efficiency vs. Quality Trade-Off and Code Quality & Maintainability Risks results show that students are very engaged in a complex and critical way. It shows that students are not just passive recipients of code, but they actively select and improve AI recommendations. This explains the diversity in considered Code Quality (CQ) scores and suggests that students' overall high acceptability is led by a powerful, reflective approach to control, a multidimensionality that quantitative surveys would fail to consider.

Our results show that students are excited to use Copilot because they find it easy and useful, and they would all suggest it. The thematic analysis shows that students put a lot of effort into figuring out if its output is the best, safest, and most suited for the design. This clearly addresses a major gap in the literature about the risks of AIPP in education, for example, how it can make programmers less able to think critically. Our findings indicate that, among this cohort, the risk may not result from passive acceptance but from the difficulty of managing the cognitive load associated with continual review. This observation agrees with Groothuijsen et. al. (2024), who emphasized that effective use of AI coding assistants depends on students' ability to critically review and validate AI-generated code rather than accepting suggestions without reflection. It also supports recent calls for integrating critical evaluation and AI-assisted code review into programming education(Zhao et al., 2026). So, the main question for computer science teaching is not whether to use AIPPs, but how to do so. Pedagogy must progress beyond fundamental tool instruction to include training in AI-assisted code review, prompt engineering for educational purposes, and the debugging of AI-generated code. In addition to the critical skills students are already adopting on their own, teachers can leverage the tool's efficiency while methodically improving code comprehension and design thinking that it may not be able to reach.

Despite the promising results, this study has several limitations. First, the study used an observational mixed-method design without a control group or pre/post-assessment. Therefore, the results should be considered evidence of students' perceptions and observed associations, rather than causal effects of GitHub Copilot on learning, code quality, or programming efficiency. Second, the evaluation was based on a single authentic software development task: implementing a responsive portfolio website. The results may not be generalizable to other programming domains such as Java, Python, algorithms, databases, or large-scale software engineering projects.

## CONCLUSION

In this study, the Technology Acceptance Model (TAM) framework was adopted to present a mixed-method evaluation of computing students' acceptance and use of the AI pair programmer GitHub Copilot. Quantitative survey data, objective performance metrics, and qualitative reflective analysis were used to provide comprehensive insight into students' perceptions of AI-assisted coding.

From a perceptual perspective, GitHub Copilot was considered exceptionally easy to use (PEOU M=4.82), highly efficient, and a significant support for programming and software development learning. This positive attitude, leading to a universal willingness to recommend the tool, indicates a low barrier to adoption of AIPP in academic settings. From performance and behavioral viewpoints, thematic analysis revealed that students did not passively consume AI output; instead, they actively critiqued it, demonstrating careful trade-offs between efficiency, code quality, security, and design. This critical behavioral engagement shows that acceptance in practice is not mere usage, but supervised collaboration.

The contribution of this work is to provide an evidence-based framework for integrating AIPPs into computing education. We move the discussion beyond the simple question of whether to adopt these tools to the pedagogical question of how. The findings demonstrate that success should not be measured by usage statistics alone, but by the quality of the human-AI collaboration it promotes.

## ACKNOWLEDGMENTS

I would like to thank the Department of Electrical Engineering, College of Engineering at Salahaddin University-Erbil, for their support and for providing facilities for the successful completion of this research study. This research also received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

## CONFLICT OF INTEREST

The author declares that he has no conflicts of interest.

## DECLARATION OF GENERATIVE AI USE

I acknowledge that Grammarly (https://www.grammarly.com) was used solely to improve grammar, readability, and language. All AI-assisted content was reviewed and edited by the authors. No AI tools were used to generate data, conduct analyses, modify research results, or draw conclusions. The authors take full responsibility for the content of the manuscript.

## DATA AVAILABILITY

The dataset generated and analyzed during the current study is available in the Mendeley Data repository: https://doi.org/10.17632/vmzrd2rjgj.2

## REFERENCES


Almeida, Y., Albuquerque, D., Filho, E. D., Muniz, F., de Farias Santos, K., Perkusich, M., Almeida, H., & Perkusich, A. (2024). AICodeReview: Advancing code quality with AI-enhanced reviews. *SoftwareX,* 26.

Andersen, N. H., Tkalich, A., Moe, N. B., Šmite, D., Söderbom, A. M., Hast, O., & Stray, V. (2026). Integrating pair programming as a work practice. *Journal of Systems and Software, 234*, 112730.

Anjaria, K. (2022). Knowledge derivation from Likert scale using Z-numbers. *Information Sciences, 590*, 234–252.

Aziz, N. (2026). GitHub Copilot in Computing Education: A Field Study Dataset. Mendeley Data.

Bañuls-Lapuerta, B., Marti-Miralles, V., Jacobo Gónzalez-García, R., Martínez-Rico, G., & Marti-Miralles, V. (2026). Using Natural Language Prompts With AI Models for Low-Cost Assistive Software Design: Exploratory Comparative Evaluation. *JMIR Rehabilitation and Assistive Technologies, 13*, 1–15.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. de O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., … Zaremba, W. (2021). Evaluating Large Language Models Trained on Code. https://arxiv.org/abs/2107.03374v2

Döderlein, J.-B., Kouadio, N. H., Acher, M., Khelladi, D. E., & Combemale, B. (2025). Piloting Copilot, Codex, and StarCoder2: Hot temperature, cold prompts, or black magic? *Journal of Systems and Software,* 112562.

Fel, S., Kozak, J., & Horodyski, P. (2025). Responsibility and AI: Exploring technology acceptance models. *Journal of Innovation and Knowledge, 10*(6).

Gəruslu, V., Jafarov, Z., Mövsümova, A., Namazov, A., & Mirzayev, H. (2026). Encouraging responsible GenAI use in software engineering education: A design-oriented model. *Journal of Systems and Software, 237*, 112846.

Ghorbian, M., Ghobaei-Arani, M., & Shakarami, A. (2026). Large language models for code generation: A survey. *Computer Standards and Interfaces, 98*.

Go, K. R., Soundarapandian, S., Mitra, A., Vidoni, M., & Ferreyra, N. E. D. (2023). Simple stupid insecure practices and GitHub's code search: A looming threat? *Journal of Systems and Software, 202*.

Gou, Q., Dong, Y., Wu, Y., & Ke, Q. (2024). RRGcode: Deep hierarchical search-based code generation. *Journal of Systems and Software, 211*.

Groothuijsen, S., van den Beemt, A., Remmers, J. C., & van Meeuwen, L. W. (2024). AI chatbots in programming education: Students' use in a scientific computing course and consequences for learning. *Computers and Education: Artificial Intelligence, 7*.

Husein, R. A., Aburajouh, H., & Catal, C. (2025). Large language models for code completion: A systematic literature review. In *Computer Standards and Interfaces (Vol. 92*). Elsevier B.V.

Jazim, F., Al-Mamary, Y. H., & Abubakar, A. A. (2025). Developing an integrated model to explore key factors influencing university students' behavioral intentions to use ChatGPT in enhancing higher education in the Hail region. *Acta Psychologica, 259*(12), 105445.

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.

Künas, C. A., Padoin, E. L., & Navaux, P. O. A. (2023). Accelerating Deep Learning Model Training on Cloud Tensor Processing Unit. International Conference on Cloud Computing and Services Science, CLOSER - Proceedings, 2023-April.

Moradi Dakhel, A., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., & Jiang, Z. M. (Jack). (2023). GitHub Copilot AI pair programmer: Asset or Liability? *Journal of Systems and Software*, 203.

Nguyen, P. T., Di Rocco, J., Di Sipio, C., Rubei, R., Di Ruscio, D., & Di Penta, M. (2024). GPTSniffer: A CodeBERT-based classifier to detect source code written by ChatGPT. *Journal of Systems and Software*, 214.

Oertel, J., Klünder, J., & Hebig, R. (2025). Don't settle for the first! How many GitHub Copilot solutions should you check? *Information and Software Technology*, 183.

Palaniappan, K., Prabhab, K. C., Veerandeswaric, M. J. & Manjula. M. (2025). Generative AI tools for accelerated software engineering. In *Advances in Computers*.

Pan, Y., Lyu, C., Yang, Z., Li, L., Liu, Q., & Shao, X. (2025). E-code: Mastering efficient code generation through pretrained models and expert encoder group. *Information and Software Technology*, 178.

Park, A., & Kim, T. (2025). Code suggestions and explanations in programming learning: Use of ChatGPT and performance. *International Journal of Management Education, 23*(2).

Rawat, A. (2021). GitHub Copilot: All you need to know | by Ayushi Rawat | Analytics Vidhya | Medium.https://medium.com/analytics-vidhya/github-copilot-all-you-need-to-know-8e6fc1d5ccc

Revuri, J., Sakthivel, R. K., & Nagasubramanian, G. (2025). Artificial intelligence (AI) technologies and tools for accelerated software development. *Advances in Computers*.

Saif, N., Khan, S. U., Shaheen, I., Alotaibi, A., Alnfiai, M. M., & Arif, M. (2024). ChatGPT: validating the Technology Acceptance Model (TAM) in the education sector via a ubiquitous learning mechanism. *Computers in Human Behavior, 154*.

Sergeyuk, A., Golubev, Y., Bryksin, T., & Ahmed, I. (2025). Using AI-based coding assistants in practice: State of affairs, perceptions, and ways forward. *Information and Software Technology, 178*, 107610.

Sîrbu, A. G., & Czibula, G. (2025a). Automatic code generation based on Abstract Syntax-based encoding. Application on malware detection code generation based on MITRE ATT&CK techniques. *Expert Systems with Applications, 264*.

Sîrbu, A. G., & Czibula, G. (2025b). Automatic code generation based on Abstract Syntax-based encoding. Application on malware detection code generation based on MITRE ATT&CK techniques. *Expert Systems with Applications, 264*.

Sorin, V., Barash, Y., Konen, E., & Klang, E. (2020). Deep Learning for Natural Language Processing in Radiology—Fundamentals and a Systematic Review. *Journal of the American College of Radiology, 17*(5).

Takerngsaksiri, W., Tantithamthavorn, C., & Li, Y. F. (2024). Syntax-aware on-the-fly code completion. *Information and Software Technology,* 165.

Tbaishat, D., AlFandi, O., Hamad, F., Bukhari, S. M. S., & Al Muhaissen, S. (2026). Modeling generative AI adoption in higher education: An integrated TAM–TPB–SDT framework with SEM validation. *Computers and Education: Artificial Intelligence*, 10.

Vlachopoulos, D., & Makri, A. (2024). A systematic literature review on authentic assessment in higher education: Best practices for the development of 21st-century skills and policy considerations. *Studies in Educational Evaluation*, 83.

Xu, S., Huang, X., Lo, C. K., Chen, G., & Jong, M. S. Yung. (2024). Evaluating the performance of ChatGPT and GPT-4o in coding classroom discourse data: A study of synchronous online mathematics instruction. *Computers and Education: Artificial Intelligence*, 7.

Yan, M., Chen, J., Zhang, J. M., Cao, X., Yang, C., & Harman, M. (2025). Robustness evaluation of code generation systems via concretizing instructions. *Information and Software Technology*, 179.

Zhao, H., Wu, Y., Lu, Z., Yu, X., Miao, W., & Chen, L. (2026). SageJavon: A scalable AI tutor for personalized programming learning. *Information Processing & Management, 63*(5), 104605.

Zhou, X., Liang, P., Zhang, B., Li, Z., Ahmad, A., Shahin, M., & Waseem, M. (2025). Exploring the problems, their causes, and solutions of AI pair programming: A study on GitHub and Stack Overflow. *Journal of Systems and Software*, 219.