Navigating the Complexity of Generative AI Adoption in Software Engineering

DANIEL RUSSO, Department of Computer Science, Aalborg University, Aalborg, Copenhagen, Denmark

This article explores the adoption of Generative Artificial Intelligence (AI) tools within the domain of software engineering, focusing on the influencing factors at the individual, technological, and social levels. We applied a convergent mixed-methods approach to offer a comprehensive understanding of AI adoption dynamics. We initially conducted a questionnaire survey with 100 software engineers, drawing upon the Technology Acceptance Model, the Diffusion of Innovation Theory, and the Social Cognitive Theory as guiding theoretical frameworks. Employing the Gioia methodology, we derived a theoretical model of AI adoption in software engineering: the Human-AI Collaboration and Adaptation Framework. This model was then validated using Partial Least Squares–Structural Equation Modeling based on data from 183 software engineers. Findings indicate that at this early stage of AI integration, the compatibility of AI tools within existing development workflows predominantly drives their adoption, challenging conventional technology acceptance theories. The impact of perceived usefulness, social factors, and personal innovativeness seems less pronounced than expected. The study provides crucial insights for future AI tool design and offers a framework for developing effective organizational implementation strategies.

CCS Concepts: • **Social and professional topics** → *Computing industry*; **Management of computing and information systems**; **Project and people management;**

Additional Key Words and Phrases: Generative AI, large language models, technology adaption, empirical software engineering

ACM Reference Format:
Daniel Russo. 2024. Navigating the Complexity of Generative AI Adoption in Software Engineering. *ACM Trans. Softw. Eng. Methodol.* 33, 5, Article 135 (June 2024), 50 pages. https://doi.org/10.1145/3652154

1 INTRODUCTION

The transformational promise of **Artificial Intelligence (AI)** is becoming increasingly evident across various sectors, with AI models demonstrating human-like competencies in areas as diverse as natural language understanding and image recognition [120]. One domain where this potential is particularly salient is software engineering, a critical function within contemporary organizations. This significance is underscored by the increasing pervasiveness of software in a broad range of products and services, with digital features enhancing their value [92]. From the initial phases of the software development lifecycle, AI tools can serve as valuable allies. Generative AI can sift through vast data sources like user feedback, market trends, and system

This work was supported by the the Danish Industry Foundation with the Sb3D project—Security by Design in Digital Denmark.
Author's address: D. Russo, Department of Computer Science, Aalborg University - Copenhagen, A.C. Meyers Vænge, 15. 2450 Copenhagen, Denmark; e-mail: daniel.russo@cs.aau.dk.

ACM 1049-331X/2024/06-ART135
https://doi.org/10.1145/3652154

logs, providing insights for feature ideation. During systems analysis and design, AI-enhanced tools can propose multiple IT architectural designs and swiftly adapt configurations, expediting the design process and product launches. In the coding phase, AI not only assists in generating code but also aids developers by crafting initial code drafts, swiftly detecting patterns, and serving as a knowledge repository. In the testing phase, AI tools can autonomously generate test cases and automate specific testing functions. During deployment, AI tools streamline the release process, ensuring that software versions are seamlessly integrated into existing systems, while also monitoring for potential deployment anomalies and facilitating rollback strategies if needed. When it comes to maintenance, insights derived from AI can aid software engineers in diagnosing issues, suggesting fixes, and predicting potential areas of improvement. The implications of AI for the field of software development could be momentous, with predictions indicating a surge in productivity ranging from 20% to 45% [19]. This substantial increase could be achieved by streamlining traditional tasks like crafting preliminary code drafts, refining existing code structures (refactoring), or conducting thorough root-cause analyses. The integration of AI not only reduces the time commitment for these activities but also enhances the overall efficiency and effectiveness of the software development process [75]. Nonetheless, despite the prospective advantages, the incorporation of language models into software engineering appears to be intricate and fraught with challenges. Indeed, there are even indications that usage of **Large Language Models (LLMs)** is on the decline, possibly as a result of end-user experimentation that found them to be ill suited to their requirements [113]. Consequently, a pressing need exists to unravel the core determinants influencing the adoption of Generative AI driven tools, such as LLM-powered tools. As we know, a diverse range of elements shape the modality and rationale behind software engineers' decision to employ language models. These incorporate both technical components, such as model quality and performance, and non-technical components, including perceived utility and ease of use [112]. Yet, there has been limited empirical research on the factors that influence language model adoption in software engineering. Hence, we formulate our research question as follows:

Research question: What influences the adoption of Generative AI tools in software engineering?

In our endeavor to explore our research question, we have applied a convergent mixed-methods approach, investigating the adoption of Generative AI and LLMs within the sphere of software engineering. To frame our understanding of AI adoption, we referenced three principal theoretical frameworks examining individual, technological, and social-level influences. These frameworks included the **Technology Acceptance Model (TAM)** [110], **Diffusion of Innovation Theory (DOI)** [85], and **Social Cognitive Theory (SCT)** [9]. By incorporating these well-validated theories, we could thoroughly comprehend the determinants of language model adoption and investigate the distinct ways these variables are operationalized in the software engineering domain. We initiated our research by conducting a questionnaire survey with a cohort of 100 software engineers. The design of these questionnaire survey was influenced by the main dimensions of our selected theories. The collected data was analyzed using the Gioia methodology [38], facilitating the development of our preliminary theoretical model. This provisional theoretical model was then validated using **Partial Least Squares–Structural Equation Modeling (PLS-SEM)**, supported by data collected from 183 software engineers. The convergence of insights derived from this comprehensive and multi-faceted investigation enhances our understanding of AI adoption within software engineering. Moreover, by understanding the adoption dynamics and impact of these disruptive technologies, this research holds the potential to guide the design of future AI tools and offer pertinent recommendations for organization-wide implementation strategies. Indeed, Generative AI tools represent a disruptive innovation in the software engineering domain, as defined by the concept of "disruptive innovation" by Christensen [18]. These tools, while

initially targeting niche applications or underserved market segments, have the potential to revolutionize traditional software engineering practices. Their ability to automate complex tasks, generate code, and offer solutions based on vast datasets challenges the status quo and can lead to a paradigm shift in how software development is approached. Over time, as these tools become more refined and widely adopted, they could displace established methodologies and tools, much like how disruptive innovations reshape industries. By understanding the adoption dynamics of such transformative technologies, this research offers insights into their potential trajectory and implications, guiding the design of future AI tools and providing recommendations for their strategic implementation across organizations.

The structure of this article unfolds as follows. In Section 2, we present a comprehensive review of related works. Our mixed-methods investigation commences with an initial theory induction, a process we detail thoroughly in Section 3. We then analyze the results of this process in Section 4. From these findings, we craft our theoretical framework in Section 5, elucidating its hypotheses. Our model undergoes a rigorous validation process using PLS-SEM, as reported in Section 6. In the concluding sections, we reflect on the broader implications and potential limitations of our study in Section 7 and sketch out future research trajectories in Section 8.

2 RELATED WORK

The software engineering landscape is undergoing a transformation with the introduction of Generative AI, especially through the capabilities of LLMs. LLMs, such as the **Generative Pre-trained Transformer (GPT)** series by OpenAI [14], owe their prowess to the foundational role of transformer architectures in **Natural Language Processing (NLP)**, which have been influential for several years [109]. Beyond their known proficiency in generating human-like text [48], LLMs, in the realm of software engineering, are poised to offer code suggestions, assist in automated documentation, aid in requirement elicitation, and more. The evolution of LLMs as Generative AI has been further propelled by the integration of transformer architectures, enhancing their understanding and generation of context [52]. As we navigate the confluence of LLMs and software engineering, it becomes evident that their potential extends beyond mere text generation. They are emerging as collaborative tools, set to redefine various facets of the software development lifecycle. Hence, throughout this article, we use the terms *Generative AI* and *LLM* interchangeably, emphasizing their relevance and potential in software engineering

A multitude of academic disciplines are currently exploring the potential implications of such advanced technologies within their respective fields. This section discusses particular context of software engineering, highlighting the prevalent themes and concerns associated with AI tools.

2.1 Assessing Generated Code: Correctness and Quality

A prominent strand of inquiry involves assessing the accuracy and quality of code generated by AI systems such as GitHub Copilot. Nguyen and Nadi [68], Dakhel et al. [27], and Yetistiren et al. [118] conducted empirical assessments to evaluate the correctness of the code generated by Copilot, finding varying degrees of success depending on the programming language and the complexity of the task. This shared focus on evaluation signifies the importance of assessing the functional integrity of the code generated by AI tools, which is a fundamental concern in software engineering.

2.2 Evaluation Criteria: Diverse Approaches

While there were commonalities in evaluating Copilot's performance, the specific aspects of evaluation varied among the studies. Nguyen and Nadi [68] focused on the performance of Copilot across different programming languages, whereas Mastropaolo et al. [62] investigated the robustness of Copilot in relation to semantic-preserving changes in the natural language description.

Yetistiren et al. [118] conducted a comprehensive assessment of the generated code in terms of validity, correctness, and efficiency. These differences underscore the multi-faceted nature of AI code generation and the various dimensions that need assessment.

2.3 Enhancing Code Productivity

Productivity in software development is positively impacted by the integration of AI tools, with tools like Copilot significantly increasing the speed of code production [49, 75]. While these tools are lauded for their productivity-enhancing capabilities, understanding their performance and limitations is vital to leveraging their potential effectively. Studies by Tian et al. [103] and Cámara et al. [15] shed light on the abilities and constraints of LLMs, such as ChatGPT. Empirical evaluation suggests that while these models demonstrate aptitude in simpler, well-structured tasks, they tend to struggle with complex tasks involving semantic nuance [103]. Additionally, a significant relationship was identified between the length of the input sequence and the quality of the output, with longer inputs often leading to poorer results [15]. These findings highlight the nuanced role of AI in software development productivity. While they enhance speed, the complexity of tasks and the length of input sequences can serve as limiting factors, pointing to areas for further improvement and optimization in these models.

2.4 Comparing Methods

Distinctly, Sobania et al. [97] embarked on a comparative study between Copilot and genetic programming, another approach in automatic program synthesis. They concluded that, despite comparable performances, genetic programming was not as mature for practical software development as Copilot. This comparative analysis provides a unique perspective on the landscape of automatic programming methodologies.

2.5 Pedagogical Concerns

Pedagogical concerns have been discussed by both Wermelinger [116] and Dakhel et al. [27], whose studies touched upon the implications of AI tools like Copilot in educational settings. Wermelinger [116] explored the implications of Copilot on teaching and assessment methods in programming courses, whereas Dakhel et al. [27] discussed the potential challenges for novice developers who might fail to filter Copilot's non-optimal solutions due to lack of expertise. These similarities highlight the significant pedagogical implications of integrating AI tools in education.

2.6 AI's Influence on the Software Development Process

Concerns around the integration and security of AI tools like GitHub Copilot are a shared finding between Jaworski and Piotrkowski [51] and Zhang et al. [119]. Both studies illustrate that despite the potential benefits of these tools, developers express hesitation and face challenges when incorporating them into their workflows, due to integration difficulties and security worries. However, these studies diverge when examining developer interactions. Zhang et al. [119] detail more practical aspects like programming languages and IDEs used with Copilot, whereas Jaworski and Piotrkowski [51] focus on the developers' sentiment toward AI tools. Ernst and Bavota [34], although also discussing the complexities of AI integration, differ by highlighting additional challenges related to legal compliance and bias. This broadens the conversation on AI's impact on software development beyond technical aspects to include ethical and legal considerations. Another commonality, albeit from a different angle, is found in the work of Bird et al. [13] and Mozannar et al. [66]. Both studies touch on the evolving role of developers as AI tools become more pervasive. Bird et al. [13] suggest a shift toward developers spending more time reviewing AI-generated code, whereas Mozannar et al. [66] provide a structured analysis of developer

interactions with AI tools, revealing inefficiencies and time costs. Thus, while the studies largely converge on the transformative potential and challenges of AI tools in software development, they also bring unique perspectives to the table, expanding the discourse to include aspects like legal concerns, workflow changes, and time costs.

2.7 Community Influence and Trust in AI Tools

The role of the community in shaping developers' trust in AI tools is investigated by Cheng et al. [16]. They present a detailed analysis of how online communities, such as forums and discussion groups, influence developers' perception of AI tools. Their research indicates that shared experiences and collective discussions play a significant role in shaping developers' trust in AI assistance.

2.8 Generative AI in Non-Coding Activities

Generative AI's impact on non-coding activities in software development is multi-faceted. A prominent aspect is the surge in productivity and creativity improvements, as noted by Ozkaya [73]. This perspective is echoed by Schmidt [93], who alludes to the potential of AI in swiftly spotting and fixing bugs. However, the two diverge in their emphasis: while Ozkaya [73] focuses on the broader paradigm shifts in software engineering conferences and research dynamics, Schmidt [93] stresses into the challenges of ensuring trustworthiness in AI systems, suggesting a complexity in their deployment. This theme of trust is further expanded upon by Ebert and Louridas [33]. They discuss the evolving nature of software systems as being more adaptive, self-modifying, and learning oriented. Contrasting this with the perspective of Ozkaya [73] on the implications of shifting to AI tools, Ebert and Louridas [33] shed light on the intricate challenges in validating such systems. They suggest that traditional software testing paradigms may no longer suffice, introducing a dimension of complexity in the deployment of AI in software development. Furthermore, the question of data quality enhancement through Generative AI offers another layer of analysis. Ebert and Louridas [33] detail the process of fine-tuning LLMs on specific datasets, emphasizing the resultant increase in output quality. This stands in contrast with the broader shifts and challenges highlighted by Ozkaya [73], focusing instead on the tactical advantages offered by AI tools. Overall, while there is a consensus on the transformative potential of Generative AI in software development, the literature also paints a picture of the challenges and nuances. From broader shifts in the research landscape to tactical advantages and validation challenges, the discourse on AI's role in non-coding activities is both rich and diverse, necessitating a holistic understanding for effective deployment.

2.9 Usability of AI Programming Assistants

The usability of AI programming assistants has been a focal point in the research, with the key motivations for usage identified as reduction in keystrokes, quick task completion, and syntax recall [58, 107]. However, developers often encounter challenges with tool control, output alignment with requirements, and difficulties with understanding, editing, and debugging generated code snippets [58, 107]. For novice programmers, cognitive and metacognitive issues arise while using these tools for assignments, indicating a need for better supportive design [77]. Additionally, developers exhibit distinct interaction modes, each requiring different forms of tool support [11]. This signifies a necessity for usability improvements in AI programming assistants, focusing on user control, cognitive effort minimization, and support for interaction modes.

In sum, the review of related work underscores the transformative potential and multi-faceted challenges posed by LLMs in the realm of software engineering. The corpus of research spans areas such as the evaluation of generated code's accuracy and quality, the augmentation of productivity, contrasting methodologies, pedagogical implications, AI's influence on software

development processes, the role of community in fostering trust, and the usability of AI programming assistants. Each study contributes uniquely to our understanding of AI's role in software engineering, highlighting the complexity of the issues at hand. While the field has made significant strides in leveraging AI's potential, the need for robust evaluation, tailored usability, and mindful integration into educational and professional settings is a recurring theme.

Beyond the aforementioned research, it is crucial to note the existence of other code generators besides Copilot. Tools such as AlphaCode [57], Amazon CodeWhisperer [6], BlackBox AI [3], CodeComplete [21], CodeGeeX [121], Codeium [22], Mutable AI [67], Ghostwriter Replit [82], and Tabnine [100] also play roles in the domain of AI-powered code generation. Interestingly, while these tools are acknowledged in the landscape, there is a glaring lack of empirical research evaluating them. Our research of the literature revealed only three papers that even mention these tools, and solely within the context of related work or discussion sections [16, 42, 114]. This presents a clear gap in the current body of research.

More specifically, while existing research thoroughly investigates the performance, usability, and impact of Generative AI tools in software engineering, it primarily focuses on the tools themselves, largely overlooking the factors influencing their adoption. A notable exception is the exploration by Cheng et al. [16] of community influence on developers' trust. Yet, this is only one facet of the broader adoption landscape, which includes individual, organizational, technological, and environmental factors. Our research question addresses this clear gap in the literature, aiming to provide a comprehensive understanding of the factors driving or hindering the adoption of these tools.

3 THEORY GENERATION

3.1 Theoretical Foundation

The multi-faceted nature of technology adoption demands the application of comprehensive theoretical frameworks that can sufficiently capture and explain the influencing factors. TAM, DOI, and SCT together offer a robust approach toward understanding the complexity of language model adoption in software engineering.

3.1.1 Technology Acceptance Model. TAM has been widely acclaimed for its relevance and efficiency in predicting and explaining the acceptance of various forms of technology [110]. Its core constructsperceived usefulness and perceived ease of useserve as an excellent starting point for understanding adoption behavior. For instance, if language models are perceived as beneficial and easy to use, software engineers are more likely to embrace them. Given its robustness and simplicity, TAM provides a foundation for understanding the fundamental determinants of technology adoption and aids in diagnosing the basic barriers to language model adoption.

3.1.2 Diffusion of Innovation Theory. While TAM primarily focuses on user perceptions, DOI complements TAM by addressing the technological characteristics influencing adoption. Rogers [85] identified key attributes of innovationsrelative advantage, compatibility, complexity, trialability, and observabilitythat significantly affect their adoption rates. As an innovation in software engineering, the acceptance of language models could be shaped by these attributes. For example, the relative advantage of language models over traditional programming methods could be a strong motivator for adoption. The compatibility of language models with existing practices and the complexity of these models might also play crucial roles. Thus, DOI adds depth to our understanding of technology-specific factors influencing language model adoption.

3.1.3 Social Cognitive Theory. The decision to adopt new technologies does not occur in a vacuum. It is influenced by the social milieu within which individuals operate. SCT comes into play

here by emphasizing the social and environmental factors that influence individual behaviors [9]. Software engineering, like any profession, has its own culture, norms, and shared beliefs that could significantly shape the adoption of language models. For instance, the prevailing norm or the extent of peer usage could encourage or discourage language model use. Moreover, the self-efficacy of individualsshaped in part by their social environmentmight affect their willingness to engage with such a new tool. SCT, therefore, adds a social layer to our understanding of language model adoption.

The selection of TAM, DOI, and SCT over other potential theories was deliberate and informed by the unique challenges and intricacies of technology adoption in the software engineering domain. While there are numerous theories available that address technology adoption, not all are equally suited to capture the nuances of language model adoption in this specific field. TAM's focus on user perceptions, DOI's emphasis on technological attributes, and SCT's consideration of the social environment together provide a holistic view that other theories might not offer in isolation. Furthermore, the combination of these three theories ensures a multi-dimensional approach, capturing the breadth and depth of factors influencing adoption. Other theories might focus too narrowly on one aspect, potentially overlooking critical influencers. By integrating these three well-established theories, we aimed to achieve a more comprehensive and nuanced understanding, ensuring that no significant factor was left unaddressed.

In summary, the triadic theoretical framework of TAM, DOI, and SCT provides a comprehensive lens to examine the adoption of language models in software engineering. By addressing the individual perceptions (TAM), technology characteristics (DOI), and social aspects (SCT), this combined framework provides a well-rounded perspective, ensuring that we cover the principal aspects influencing the decision to adopt language models. This choice of theories allows us to glean insightful details that not only offer a rich understanding of the current adoption scenario but also inform strategies to expedite future adoption.

3.2 Questionnaire Survey Guideline

Our investigation covers three units of analysis: individual-level factors, technology-level factors, and social-level factors, inspired by TAM, DOI, and SCT. We aim to reveal a comprehensive understanding of the acceptance and use of LLM-powered tools in the software engineering context. Here, we detail the final questionnaire survey questions designed to capture these constructs effectively. As a preliminary step, we conducted a pilot interview with a senior engineering manager from a prominent software company based in Central Europe to ensure the clarity and appropriateness of our questions in April 2023. To design this investigation, we used the SIGSOFT Empirical Standard for Questionnaire Surveys [79].

Individual-level factors, derived from TAM, focus on perceived usefulness, perceived ease of use, and behavioral intention:

— *Perceived usefulness*: "To what extent do you think using language models increases your efficiency as a software engineer?" This question is designed to gauge how software engineers perceive the potential productivity gains from using LLMs.
— *Perceived ease of use*: "How easy do you think it is to learn how to use a new language model effectively?" This question aims to capture the perceived cognitive effort required to learn and adapt to LLMs.
— *Behavioral intention*: "How likely are you to use a language model in your work in the next six months? And for which tasks?" These questions aim to evaluate the intention of software engineers to adopt LLMs in their near-future tasks.

Technology-level factors, drawing from the diffusion of innovations, include compatibility, relative advantage, and complexity:

— *Compatibility*: "How is using a language model different from your current software engineering practices?" This question assesses the perceived fit of LLMs with existing practices and workflows.
— *Relative advantage*: "What potential benefits do you think language models offer over your current methods?" This question helps identify the perceived benefits of LLMs compared to traditional methods.
— *Complexity*: "What concerns do you have about using a language model in your work?" This question aims to highlight any perceived barriers or challenges associated with LLM adoption.

Social-level factors, grounded in SCT, encompass social influence, environmental factors, and self-efficacy:

— *Social influence*: "How much do your colleagues or peers influence your decisions to use language models?" This question examines the impact of social norms and colleagues' opinions on the acceptance of LLMs.
— *Environmental factors*: "In your opinion, to what extent do you feel your organization is supportive of adopting language models as a standard technology?" This question explores the role of organizational support in fostering LLM adoption.
— *Self-efficacy*: "How important is it to you to be seen as someone who uses cutting-edge technology in your work?" This question aims to capture an individual's self-confidence in their ability to use advanced technologies like LLMs effectively.

Through these questionnaire survey questions, we strive to understand the complex interplay of individual, technological, and social factors that contribute to the adoption and usage of LLM-powered tools among software engineers.

3.3 Participants

The data collection was executed via Prolific Academic [74], a well-regarded academic data collection platform often utilized by the software engineering community [28, 88–90]. We solicited the input of 100 software engineers, who were asked to answer a series of nine open-ended questions founded on theoretical principles. The compensation for participants (10 minutes) exceeded the U.S. federal minimum wage.[1] The survey was conducted on the Qualtrics platform.

Participants were meticulously chosen through a two-step screening process. Initially, a pre-screening phase was conducted where participants were filtered based on specific self-reported characteristics, including proficiency in computer programming, full-time employment in the software industry, a negative student status, a degree in computer science, and a 100% approval rate. Following this, a competence screening was performed, as per the methods described by Danilova et al. [28]. This second screening involved assessing participants' knowledge and understanding in key areas, including compilers, programming aid websites, and recursive functions. Furthermore, professionals affirmed their familiarity with Generative AI tools and confirmed their use to a certain extent.

Our data collection methodology complied strictly with the ethical guidelines of the Declaration of Helsinki [71]. The Research Ethics Committee at Aalborg University approved this research project in March 2023. All participants were older than 18 years, gave informed consent prior to

[1]We consistently adhered to the suggested compensation by Prolific. For your reference, participants were paid $9.00 per hour for their time (i.e., $1.5 for this investigation).

participating in the study, and were notified of their right to withdraw their participation at any point. Additionally, the authors have completed formal training in research ethics for engineering and behavioral sciences.

In terms of participant demographics, men made up 76% of the sample, women accounted for 23%, and non-binary individuals represented 1%. Geographically, the participants came from various regions: Portugal (23%), South Africa (15%), Italy (10%), United Kingdom (9%), Poland (9%), and other countries (34%).

The professional experience of the participants ranged across various stages in the software industry: 26% had 0 to 1 year of experience, 54% had 2 to 3 years, 9% had 4 to 5 years, 9% had 6 to 10 years, and 2% had more than 10 years of experience.

As for their roles in the industry, the majority were software developers or programmers (83%). This was followed by testers or QA engineers (7%); data analysts, data engineers, or data scientists (5%); team leads (3%); and UX/UI designers (2%).

3.4 Analysis of the Qualitative Data

Data analysis was implemented within the naturalistic inquiry paradigm [59] context, complemented by the constant comparison method [40]. The crucial role these strategies play in qualitative data acquisition and examination is significant. This iterative process facilitates initial theory development by identifying patterns and broader dimensions [39], derived from continual data comparison and analysis, and refining it in accordance with the participants' input [50].

A thematic analysis approach was utilized to process the data. Thematic analysis is a commonly employed method in qualitative research, which involves identifying, analyzing, and reporting patterns or themes within the data, while providing a rich, detailed, and complex account of the data [20]. The structured methodology proposed by Gioia et al. [38] served as the analytical framework. Recent trends within the Management Science community have seen the adoption of this methodology, emphasizing its potential in reinforcing scientific rigor [44, 60]. The approach is structured and dedicated to encouraging comprehensive theoretical progression [38].

The Gioia methodology segments data processing into three stages. The inaugural stage revolves around recognizing *first-order concepts*, or *in vivo* codes [99], which align closely with the participants' own words, with minimal researcher-imposed categorization. These codes were then collated into broader themes, a process known as open coding [108].

In the subsequent stage, similarities and differences are identified, and emergent themes from these comparisons contribute to the explanation and depiction of the phenomena under investigation. We explored the associations between the concepts to create our high-level themes, employing axial coding. These are the *second-order themes*.

The final stage amalgamates similar second-order themes into *aggregate dimensions*, representing the apex of theoretical contribution. This process was iterative and process oriented [61], and was perpetuated until theoretical saturation was accomplished [25].

The outcome of this investigation is the *data structure*, which encapsulates first-order terms, second-order themes, and aggregate dimensions for each of the nine theoretical dimensions of our investigation. Notably, the aggregate dimensions were not pre-conceived categories defined prior to the analysis; rather, they are the end product of a refined and iterative analytical process.

For example, in Table 1, which details the data structure of perceived sefulness of LLMs in software engineering, a clear progression from first-order concepts to second-order themes and aggregate dimensions is demonstrated. Consider the quote from participant R-15, which is categorized under first-order concepts as "Automating tasks, reducing time and effort, manual coding, documentation." These concepts are then synthesized into the second-order theme of "Task-specific efficiency improvements," indicating a broader category of improvements in

Table 1. Data Structure of Perceived Usefulness of LLMs in Software Engineering

ID	Quote	1st-Order Concepts	2nd-Order Themes	Aggregate Dimensions	(%)
R-15	The can greatly increase my efficiently by automating certain tasks and reducing the time and effort it take for manual coding and documentation	Automating tasks, reducing time and effort, manual coding, documentation	Task-specific efficiency improvements	Efficiency Improvement	55
R-28	it increases considerably my efficiency specially in simple tasks	Increased efficiency, simple tasks	Task-specific efficiency improvements	Efficiency Improvement	55
R-52	They help only in monotonous and simple tasks (defining constructors and writing user input validating systems for example).	Monotonous tasks, simple tasks	Task-specific efficiency improvements	Efficiency Improvement	55
R-67	I save 10% - 20% of time	Time savings	Time savings	Time Savings	24
R-30	It certainly helps a lot, these last few days that I've been particularly using ChatGPT (with GPT 3-5), my efficiency has gone up by quite a bit. It helps me debug code faster, learn about new features without scanning the whole documentation, and providing me useful code snippets for my work.	Increased efficiency, debugging, learning new features, code snippets	Task-specific benefits, learning enhancement	Task-Specific Benefits	26
R-17	I think language models can increase the efficiency of software engineers by automating certain tasks, such as code generation, testing, and documentation. Additionally, language models can help with data analysis and decision-making, allowing engineers to make informed choices based on large datasets. Overall, language models have the potential to enhance productivity and efficiency in software engineering, but they should be used as a tool alongside human expertise and judgement.	Automating tasks, data analysis, decision making, human expertise, judgment	Complementary tool, efficiency improvement	Complementary Tool	18
R-21	Very slightly. It's nice for generic tasks, but the models have zero knowledge about our internal APIs so they're really hard to apply.	Limited applicability, internal APIs, generic tasks	Limited applicability	Limited Applicability	15
R-41	I'm much more efficient using a language model as it have been helping me to understand the company's code much faster.	Increased efficiency, understanding company code, faster learning	Learning enhancement	Learning Enhancement	12
R-76	While most of the time, language models get small things wrong, thereby requiring extra time for checking their output, the time they save by doing especially the boring parts of coding for you definitely outweighs this in my opinion. I would say using copilot for example has increased my coding efficiency by about 30%.	Time savings, checking output, increased efficiency	Time savings, quality concerns	Time Savings, Quality Concerns	24,9
R-55	I think it helps but then you have to review and understand the code anyway. So I don't know if it makes you more efficient.	Code review, understanding, efficiency concerns	Quality concerns	Quality Concerns	9

efficiency related to specific tasks. Subsequently, this second-order theme is aggregated into the dimension of "Efficiency Improvement," representing a generalized area of impact in software engineering through the use of LLMs. This example illustrates the analytical progression from specific participant quotes to broader thematic categories, demonstrating the methodology of our study.

The presentation of the data structure with their respective second-order themes and first-order concepts are reported in Tables 1 through 9.

4 RESULTS

4.1 Perceived Usefulness of LLMs in Software Engineering

TAM has been widely used in the study of technology adoption, focusing on two key predictors of acceptance: perceived usefulness and perceived ease of use [29]. In the context of software engineering, the perceived usefulness of LLMs can be examined by evaluating how they contribute to efficiency, productivity, and performance enhancement. Table 1 provides a summary of software engineers' perceptions of LLMs in their work.

4.1.1 Efficiency Improvement. One of the main perceived benefits of LLMs is their ability to improve efficiency. As shown in Table 1, efficiency improvement is the most frequently mentioned aggregate dimension (55%). Engineers recognize that LLMs can automate certain tasks, reduce time and effort, and simplify monotonous tasks. For example, R-15 highlighted that LLMs can "increase my efficiency by automating certain tasks and reducing the time and effort it takes for manual coding and documentation." This finding aligns with the TAM's emphasis on perceived usefulness, which posits that users will adopt technology if they perceive it to be useful in enhancing their performance [29].

4.1.2 Task-Specific Benefits. Another aspect of perceived usefulness is the task-specific benefits LLMs provide, such as debugging, learning new features, and generating code snippets. As R-30 mentioned, LLMs have significantly increased their efficiency by helping them "debug code faster, learn about new features without scanning the whole documentation, and providing useful code snippets for work." This category represents 26% of the aggregate dimensions and supports the notion that perceived usefulness is an important predictor of LLM adoption [111].

4.1.3 Complementary Tool. LLMs are also viewed as a complementary tool to human expertise and judgment. R-17 pointed out that "language models have the potential to enhance productivity and efficiency in software engineering, but they should be used as a tool alongside human expertise and judgment." This perception highlights the importance of balancing the benefits of LLMs with the need for human oversight, an aspect that may influence the overall perceived usefulness of the technology.

4.1.4 Limited Applicability and Quality Concerns. While many respondents reported positive perceptions of LLMs, some expressed concerns about their limited applicability (15%) and quality concerns (9%). For instance, R-21 mentioned that LLMs are "nice for generic tasks, but the models have zero knowledge about our internal APIs so they're really hard to apply." R55 also noted that while LLMs may help, "you have to review and understand the code anyway. So I don't know if it makes you more efficient." These concerns suggest that while LLMs can offer benefits in certain situations, their usefulness may be limited by the need for review and adaptation to specific contexts. This finding is consistent with the TAM literature, which highlights that the perceived usefulness of a technology is not only determined by its benefits but also by its limitations [111].

Table 2. Data Structure of Perceived Ease of Use of LLMs in Software Engineering

ID	Quote	1st-Order Concepts	2nd-Order Themes	Aggregate Dimensions	(%)
R-46	easy as you practice more	Practice, improvement	Practice and improvement	Learning Process	54
R-4	it takes some time and dedication	Time, dedication	Learning curve	Learning Process	54
R-96	However, for more advanced tasks, it can take some time to word your questions correctly	Advanced tasks, time	Learning curve	Learning Process	54
R-30	In the first week of using it, you will be already increasing your efficiency	Efficiency, time	Practice and improvement	Learning Process	54
R-33	To be more efficient, it requires to learn the specific prompts that give you an exact answer you expect	Efficiency, specific prompts	Practice and improvement	Learning Process	54
R-17	depends on the complexity and capabilities of the model, as well as the user's prior experience and knowledge	Complexity, user experience, prior knowledge	Individual factors	Individual Background	26
R-98	It is easy when you know another language that is similar to it	Prior knowledge, similar language	Individual factors	Individual Background	26
R-5	Extremely easy	Ease of use	Perceived ease of use	Perceived Ease of Adoption	13
R-52	Easy to use, hard to master the prompts	Ease of use, mastery	Mastery	Perceived Ease of Adoption	13
R-74	The difficult part is to understand if the response given is correct and related to what someone needs	Response quality, relation to needs	Perceived difficulty	Model Effectiveness	6

The perceived usefulness of LLMs in software engineering, as reflected in the efficiency improvement, task-specific benefits, and complementary nature of the technology, supports the potential for widespread adoption. However, the concerns related to limited applicability and quality highlight the importance of addressing these limitations to enhance the perceived usefulness and, consequently, the acceptance of LLMs. This analysis aligns with the TAM framework, which emphasizes that perceived usefulness is a critical determinant of technology acceptance [29].

4.2 Perceived Ease of Use of LLMs in Software Engineering

In this sub-section, we present the results of our qualitative analysis of the questionnaire survey statements, highlighting the key factors that influence the perceived ease of use of LLMs in software engineering. Our analysis draws upon the TAM framework [29], which posits that the perceived ease of use and perceived usefulness are essential determinants of technology adoption. We have identified several aggregate dimensions that explain the perceived ease of use of LLMs and present them in the following, providing empirical evidence from the questionnaire survey statements (Table 2).

4.2.1 Learning Process. Our analysis reveals that the learning process is a crucial factor influencing the perceived ease of use of LLMs in software engineering. As shown in Table 2, R-54 reported that "once I got familiar with the technology, it became much easier to use." This finding aligns with TAM proposed in the work by Davis [29], which suggests that the perceived ease of use of a technology is directly related to its adoption. Moreover, prior research has emphasized

the role of learning in the adoption of new technologies [110]. In this context, the learning curve associated with LLMs appears to be an essential determinant of their perceived ease of use.

4.2.2 Prior Experience. The questionnaire survey data also underscore the importance of prior experience in shaping the perceived ease of use of LLMs. For example, R-20 stated, "I think it is easy when you know the different concepts." This observation is consistent with the literature on technology adoption, which suggests that individuals with prior experience in related technologies are more likely to perceive a new technology as easy to use [24]. In the case of LLMs, having a background in programming languages or NLP could facilitate their adoption in software engineering.

4.2.3 Individual Differences. Another key theme that emerged from our analysis is the role of individual differences in shaping the perceived ease of use of LLMs. As R-9 noted, "it depends on the person and how they are used to work." This finding supports the notion that individual characteristics, such as cognitive style and personal innovativeness, can influence the perceived ease of use of a technology [94]. In the context of LLMs, the extent to which software engineers perceive them as easy to use may depend on their unique preferences, learning styles, and problem-solving approaches.

4.2.4 Intuitiveness and User Interface. The intuitiveness of LLMs and their user interface design also emerged as important factors in our analysis. For instance, R-89 mentioned that "they were pretty much made for ease of use by the average consumer." This observation aligns with the work of Nov and Ye [69], who argued that a well-designed user interface can significantly enhance the perceived ease of use of a technology. In the case of LLMs, an intuitive and user-friendly interface could facilitate their adoption among software engineers.

4.2.5 Task Complexity. Finally, the complexity of the tasks that LLMs are used for in software engineering appears to influence their perceived ease of use. As R-53 noted, "the difficulty to learn how to use them effectively can vary as it depends on how you're using it, but for the most part, it ranges from not too hard to very hard." This finding is consistent with the Task-Technology Fit model [41], which posits that the fit between the technology and the task it is intended for affects the technology's perceived ease of use and, ultimately, its adoption. In the context of LLMs, it seems that software engineers may find them easier to use for certain tasks, whereas others might require a higher level of expertise and knowledge.

In summary, our analysis identified several aggregate dimensions that explain the perceived ease of use of LLMs in software engineering, including the learning process, prior experience, individual differences, intuitiveness and user interface, and task complexity. These factors provide a nuanced understanding of the adoption of LLMs in software engineering and their connection to the theoretical framework of TAM. By incorporating the empirical evidence from the questionnaire survey statements and drawing on relevant literature, our findings contribute to the ongoing conversation about the role of LLMs in software engineering and the factors that influence their adoption.

4.3 Behavioral Intention of LLMs in Software Engineering

TAM has been widely used to understand the factors influencing the adoption of new technologies in various contexts, such as software engineering [30, 110]. According to TAM, behavioral intention, which reflects the likelihood of an individual to use a specific technology, is influenced by two main factors: perceived usefulness and perceived ease of use [30]. In this sub-section, we explore the behavioral intention of software engineers in relation to the adoption of LLMs, focusing on the aggregate dimensions emerged from the performed analysis (Table 3).

Table 3. Data Structure of Behavioral Intention of LLMs in Software Engineering

ID	Quote	1st-Order Concepts	2nd-Order Themes	Aggregate Dimensions	(%)
R-8	Very likely. I am considering purchasing a ChatGPT-4 subscription, mainly to refactor (legacy) code or to make it adhere to certain design patterns. It could also help refactoring code to make it more SOLID.	Refactor code, design patterns, SOLID principles	Code generation and refactoring	Code Improvement and Maintenance	42
R-35	I believe I may start using a language more and more, especially to automate tasks which can be performed by a language model and which take a significant amount of time.	Automate tasks, time saving	Task automation and optimization	Efficiency and Automation	35
R-25	Very likely. Mostly to find documentation for libraries, refactor and clarify confusing code.	Find documentation, refactor code, clarify confusing code	Information seeking and learning	Learning and Problem Solving	28
R-7	Tried it out with some basic programming related questions already.	Basic programming questions	Information seeking and learning	Learning and Problem Solving	28
R-62	Very likely. For writing basic functionalities, defining tasks, for emails	Writing basic functionalities, defining tasks, e-mails	Task-specific use cases	Specialized Applications	20
R-17	Very likely in language translation, text summarization and text generation	Language translation, text summarization, text generation	NLP tasks	NLP and Content Generation	18
R-46	Writing automated tests	Writing automated tests	Testing and code validation	Quality Assurance and Validation	15
R-69	Not likely	Non-adoption	Uncertainty or non-adoption	Adoption Barriers and Concerns	12
R-60	The cost and dependency on a third-party service might be a concern.	Cost, dependency on third-party service	Adoption barriers	Adoption Barriers and Concerns	12

4.3.1 Code Improvement and Maintenance. A significant portion of software engineers indicated their intention to use LLMs to improve and maintain their codebase. This finding aligns with TAM's concept of perceived usefulness, as using LLMs for code refactoring, adherence to design patterns, and implementation of SOLID principles can enhance software quality and maintainability [76]. For instance, R-8 mentioned, "I am considering purchasing a ChatGPT-4 subscription, mainly to refactor (legacy) code or to make it adhere to certain design patterns. It could also help refactoring code to make it more SOLID." This quote highlights the potential value of LLMs in addressing common software engineering challenges.

4.3.2 Efficiency and Automation. LLMs were perceived to be useful for automating repetitive tasks and increasing efficiency. This perception corresponds to both perceived usefulness and perceived ease of use in TAM, as task automation can lead to time savings and streamline the development process [110]. R-35 stated, "I believe I may start using a language model more and

more, especially to automate tasks which can be performed by a language model and which take a significant amount of time." The adoption of LLMs for task automation can potentially improve software engineers' productivity.

4.3.3 Learning and Problem Solving. The use of LLMs for learning and problem solving was another theme identified, which is in line with TAM's perceived usefulness. Software engineers expressed the intention to use LLMs for tasks such as finding documentation, clarifying confusing code, and seeking information on programming-related questions. As R-25 stated, "Very likely. Mostly to find documentation for libraries, refactor and clarify confusing code." LLMs can serve as a valuable learning and problem-solving tool for software engineers, supporting continuous professional development.

4.3.4 Specialized Applications. Respondents mentioned the potential use of LLMs for specific tasks, such as writing basic functionalities, defining tasks, and composing e-mails. This theme is related to the perceived usefulness of LLMs in addressing particular software engineering needs. R-62, for example, mentioned, "Very likely. For writing basic functionalities, defining tasks, for emails." The adoption of LLMs for specialized applications can provide targeted benefits to software engineers in their daily work.

4.3.5 Adoption Barriers and Concerns. Despite the potential benefits of LLMs, some software engineers expressed concerns and barriers to adoption, such as cost, dependency on third-party services, and potential ethical issues. These concerns align with TAM's concept of perceived ease of use, as they can hinder the adoption of LLMs [110]. For instance, R-60 stated, "The cost and dependency on a third-party service might be a concern." Understanding and addressing these concerns is essential for promoting LLM adoption in software engineering.

In conclusion, our analysis reveals several factors that influence the behavioral intention of software engineers to adopt LLMs, aligning with the theoretical framework of TAM. LLMs are perceived as useful for code improvement and maintenance, efficiency and automation, learning and problem solving, and specialized applications, although some adoption barriers and concerns persist. By understanding these factors, we can better support the integration of LLMs into software engineering practices and promote their adoption to enhance productivity and software quality.

4.4 Compatibility of LLMs in Software Engineering

The compatibility of LLMs in software engineering is a crucial factor in understanding their adoption and impact on software development practices. Compatibility refers to the degree to which an innovation is perceived as being consistent with existing values, past experiences, and the needs of potential adopters [85]. This sub-section presents a thematic analysis of the compatibility of LLMs in software engineering based on the responses of software engineers, which are summarized in Table 4. We have identified four aggregate dimensions, as detailed in Sections 4.4.1 through 4.4.4.

4.4.1 Improved Efficiency. Improved Efficiency was the most frequently occurring theme in the data, with 39% of the responses reflecting this aspect of compatibility. The use of LLMs in software engineering tasks is perceived to speed up the development process, automate mundane tasks, and ultimately improve overall efficiency. One respondent (R-19) highlighted that LLMs "can be used to automate certain tasks in software engineering," thus reducing the time and effort spent on repetitive tasks. Similarly, R-35 noted that using a language model "will speed up the tasks of going to look for specific pieces of code from multiple websites." These findings align with the literature, where compatibility is a key factor for technology adoption [105].

Table 4. Data Structure of Compatibility of LLMs in Software Engineering

ID	Quote	1st-Order Concepts	2nd-Order Themes	Aggregate Dimensions	(%)
R-19	Language models can be used to automate certain tasks in software engineering . . .	Automate tasks, speed up process	Improved software engineering tasks	Improved Efficiency	39
R-27	It shifts the burden of research from me to the algorithm.	Shift burden, algorithm	Improved software engineering tasks	Improved Efficiency	39
R-35	A language model will speed up the tasks of going to look for specific pieces of code from multiple websites.	Speed up tasks, code search	Improved software engineering tasks	Improved Efficiency	39
R-1	It gives an extra hand and can provide assistance for things I don't know and can't find with a traditional search.	Extra hand, assistance	Providing help and support	Assistance and Support	28
R-15	There is not much of a difference as the language model is just used to assist my current software engineering practices.	Similarity, assistance	Similarity to current practices	Similarity to Current Practices	16
R-5	Not that different, it's like Googling but I don't need to filter as much information.	Similarity, less filtering	Similarity to current practices	Similarity to Current Practices	16
R-46	It's something new: you have to learn how to use it.	Learning, adaptation	Need for adaptation and learning	Adaptation and Learning	11
R-87	Using a language model represents a new paradigm for me . . .	New paradigm, adaptation	Need for adaptation and learning	Adaptation and Learning	11
R-99	Far more interactive and personalized compared to normal google searches.	Interactive, personalized	Personalized and interactive use	Personalization and Interaction	10
R-58	It offers a more specific and tailored response compiled from a lot of content available online . . .	Specific, tailored response	Personalized and interactive use	Personalization and Interaction	10

4.4.2 Assistance and Support. Assistance and Support emerged as the second most frequent theme in the data, representing 28% of the responses. Respondents highlighted the value of LLMs in providing help and support, particularly in situations where traditional search methods fail to deliver desired results. R-1 mentioned that LLMs can provide an "extra hand and assistance for things I don't know and can't find with a traditional search." This demonstrates that LLMs' ability to offer contextually relevant and targeted assistance is seen as an important aspect of their compatibility with software engineering practices.

4.4.3 Similarity to Current Practices. Similarity to Current Practices was reported by 16% of the respondents. This theme suggests that the adoption of LLMs in software engineering is facilitated by their perceived similarities to existing tools and practices. R-15 noted that there is "not much of a difference as the language model is just used to assist my current software engineering practices." R-5 similarly stated that using LLMs is "like Googling but I don't need to filter as much information." The perceived similarity to current practices can influence the adoption of LLMs, as it reduces the barriers to their integration into existing workflows [85, 105].

4.4.4 Adaptation and Learning. Adaptation and Learning was reported by 11% of the respondents. This theme highlights the importance of learning and adapting to new tools and techniques in the software engineering domain. R-46 expressed that using LLMs is "something new: you have to learn how to use it." Similarly, R-87 mentioned that using a language model "represents a new paradigm for me." This theme indicates that the compatibility of LLMs in software engineering can be enhanced by promoting learning and adaptation among software engineers. The adoption of LLMs can thus be facilitated by providing resources and training to help engineers understand and integrate these tools into their daily work.

In conclusion, this sub-section has explored the compatibility of LLMs in software engineering by analyzing the aggregate dimensions derived from the responses of software engineers. These dimensionsImproved Efficiency, Assistance and Support, Similarity to Current Practices, and Adaptation and Learningprovide a comprehensive understanding of the factors that influence the compatibility of LLMs in software engineering. By linking these dimensions to DOI [85], this analysis offers valuable insights into the factors that contribute to the adoption and integration of LLMs into software engineering practices. This understanding can help inform the development of LLMs that are more compatible with existing workflows and practices, ultimately leading to more widespread adoption and use in the software engineering domain.

4.5 Complexity of LLMs in Software Engineering

The complexity of adopting LLMs in software engineering is a critical aspect of understanding their diffusion and impact on the industry. According to DOI, the complexity of an innovation influences its adoption rate, with more complex innovations facing a slower adoption [85]. The thematic analysis of the questionnaire survey data (Table 5) reveals several aggregate dimensions that contribute to the perceived complexity of LLMs in software engineering. In this sub-section, we discuss each of these dimensions and their implications for the adoption of LLMs, linking them to the relevant literature and DOI.

4.5.1 Job Security Concerns. The fear of job loss and skill devaluation emerged as a significant concern among respondents (25% frequency). Respondent R-15 stated, "I'm concerned that it can automate a lot of tasks and make most of my work obsolete." This perspective aligns with research on the potential disruptive effects of AI and automation on the job market [35]. According to DOI, innovations that are perceived to threaten job security are likely to face resistance [85]. To mitigate this challenge, organizations should communicate the benefits of LLMs and focus on upskilling and reskilling employees [12, 101].

4.5.2 Dependence and Complacency. Some respondents (16% frequency) expressed concerns about junior programmers relying too much on LLMs, leading to a decline in code understanding and increased reliance on these models. Respondent R-34 explained, "My concern is other junior programmers using it without understanding the code and causing bugs (more work for me)." This challenge can be addressed by promoting the responsible use of LLMs and ensuring that programmers have a strong foundation in coding concepts.

4.5.3 Data Security and Privacy. Data security and privacy concerns were identified by 15% of respondents. They expressed concerns about LLMs being trained on sensitive data, potentially leading to privacy breaches. Respondent R-17 mentioned, "In terms of privacy, as language models can be trained on sensitive or personal data, such as emails, messages, or documents. This may raise privacy and data protection concerns." To address this issue, developers should ensure that LLMs are trained on secure, anonymized datasets and that privacy regulations are followed.

Table 5. Data Structure of Complexity of LLMs in Software Engineering

ID	Quote	1st-Order Concepts	2nd-Order Themes	Aggregate Dimensions	(%)
R-15	I'm concerned that it can automate a lot of tasks and make most of my work obsolete.	Automation, task replacement, obsolescence	Job loss, skill devaluation	Job security concerns	25
R-34	My concern is other junior programmers using it without understanding the code and causing bugs (more work for me).	Junior programmers, lack of understanding, bugs	Decline in code understanding, reliance on language models	Dependence and complacency	16
R-17	In terms of privacy, as language models can be trained on sensitive or personal data, such as emails, messages, or documents. This may raise privacy and data protection concerns, especially if the language model is used in a cloud-based environment or shared with third parties.	Privacy, data protection, sensitive data	Data confidentiality, exposure risk	Data security and privacy	15
R-49	Language models aren't perfect, so I would be afraid that they would cause errors.	Language model imperfection, errors	Incorrect or poorly maintained code	Quality and accuracy of generated code	13
R-28	Author rights are tricky to attribute	Authorship, rights	Legal concerns	Ethical and legal considerations	8
R-87	Concerns I have about using language models in my work include issues of bias, explainability, and potential security vulnerabilities.	Bias, explainability, security vulnerabilities	Trust in generated results, interpretability	Bias and explainability	7
R-6	Might not be compatible with the systems at my workplace.	Compatibility, systems	Integration challenges	Compatibility and integration	5
R-5	My programming skills might get "rusty"if I rely too much on it, like with autocorrect, I feel my grammar is now worse because my phone understands even incomplete words and I can't remember the proper way of writing some words because of this.	Relying on language models, skill deterioration, autocorrect	Personal skill decline	Skill deterioration	3

4.5.4 Quality and Accuracy of Generated Code. Respondents also raised concerns about the quality and accuracy of code generated by LLMs (13% frequency). Respondent R-49 remarked, "Language models aren't perfect, so I would be afraid that they would cause errors." Ensuring the reliability and accuracy of generated code is essential for LLM adoption [81]. To address this issue, developers should establish best practices for code review and validation, as well as invest in improving the models' performance [115].

4.5.5 Ethical and Legal Considerations. Ethical and legal considerations, such as authorship and intellectual property rights, were identified by 8% of respondents. Respondent R-28 simply stated, "Author rights are tricky to attribute." Organizations should consider the ethical implications of using LLMs and establish guidelines for their use to ensure compliance with existing laws and regulations [64].

4.5.6 Bias and Explainability. Some respondents (7% frequency) highlighted concerns about the potential biases in outputs and the lack of explainability in their decision-making processes. Respondent R-62 expressed, "The biases in the model can have unintended consequences." It is essential to address these issues to ensure the responsible use of LLMs in software engineering. Organizations can invest in research to reduce biases and improve the explainability of LLMs to enhance their trustworthiness and adoption [7].

4.5.7 Integration and Compatibility. Integration and compatibility issues were mentioned by 6% of respondents, who expressed concerns about the ability of LLMs to work seamlessly with existing software development tools and practices. Respondent R-21 stated, "Integrating the model into the current workflow might be challenging." To facilitate the adoption of LLMs, developers should ensure that these models are compatible with existing tools and can be easily integrated into the software development process.

In conclusion, the complexity of LLM adoption in software engineering is multi-faceted, encompassing various concerns, such as job security, dependence, data security, code quality, ethical issues, bias, and integration challenges. By addressing these concerns, organizations can facilitate the adoption of LLMs and leverage their potential benefits in software engineering. This discussion, grounded in the thematic analysis of the questionnaire survey data and DOI, contributes to the understanding of the factors that affect LLM adoption in the software engineering context.

4.6 Relative Advantage of LLMs in Software Engineering

The relative advantage of LLMs in software engineering is a key factor in their adoption, as postulated by DOI [85]. Our qualitative data analysis, based on Gioia's methodology, highlights the various aspects of LLMs that contribute to their perceived benefits over current methods. In the following, we present the Aggregate Dimensions derived from our analysis and discuss their implications in relation to the relative advantage construct (Table 6).

4.6.1 Time Efficiency. A recurring theme in our analysis was the time efficiency provided by LLMs, as reported by 42% of respondents. Respondents appreciated the ability of LLMs to quickly complete tasks, search for relevant information, and provide solutions to coding issues. For example, R-25 noted that LLMs significantly reduced time spent searching for information: "I believe the best thing is the time spent searching for certain things is way lower than before." This time efficiency can be attributed to the NLP capabilities of LLMs, which enable users to communicate their needs more effectively and rapidly obtain tailored solutions [14].

4.6.2 Code Quality. Improvements in code quality emerged as another key advantage of LLMs, with 14% of respondents highlighting the positive impact on their work. Respondents reported that LLMs provided clearer, more understandable code, which reduced errors and improved overall code robustness. R-37 stated: "It generates a simpler and more understandable code, working in a more organized way and reducing errors." This improvement in code quality is a result of LLMs' ability to analyze and learn from vast amounts of source code, allowing them to provide optimal solutions based on best practices [31].

4.6.3 User Experience. LLMs were noted to enhance the overall user experience, with 11% of respondents mentioning the ease of use and communication with the models. R-67 commented on the human-like interaction: "Language models are easier to use and faster because you can send messages like for human." The improved user experience can be attributed to the natural language understanding capabilities of LLMs, which allow them to interpret and respond to user inputs more effectively than traditional methods [78].

Table 6. Data Structure of Relative Advantage of LLMs in Software Engineering

ID	Quote	1st-Order Concepts	2nd-Order Themes	Aggregate Dimensions	(%)
R-25	I believe the best thing is the time spent searching for certain things is way lower than before.	Time-saving search	Task completion	Time Efficiency	42
R-37	It generates a simpler and more understandable code, working in a more organized way and reducing errors.	Understandable code, reduced errors	Code simplicity and robustness	Code Quality	14
R-67	Language models are easier to use and faster because you can send messages like for human	Easier to use, faster	Ease of communication	User Experience	11
R-23	I think with language models you don't have to spend that much time learning 'technical' things	Less time learning	Simplified learning	Learning and Skill Development	9
R-64	It can help me jump over boilerplate code and snippets I typed hundreds of times before. It can also help in learning new languages' syntax.	Boilerplate code, learning new languages	Code generation and automation	Automation and Adaptability	8
R-35	I believe it might be more efficient and save some time.	Efficient, time saving	Innovative solutions	Creativity and Innovation	3
R-11	I think it can bring clear points of view to the table that weren't take into consideration	New points of view	Diverse perspectives	Insights and Decision <aking	4
R-68	More effective and faster problem resolution	Faster problem resolution	Effective troubleshooting	Problem Solving	6
R-53	Language models (paired programming) can help make fewer mistakes and overall increase efficiency.	Fewer mistakes, increased efficiency	Collaborative assistance	Teamwork and Collaboration	5
R-94	It gives more digested information, and we can "mold" the information how we want (e.g. asking the language model to respond using short setences, or to explain in detail certain topics, etc)	Digestible information, customization	Flexible responses	Customization and Personalization	8

4.6.4 Learning and Skill Development. LLMs were also found to facilitate learning and skill development, as reported by 9% of respondents. Respondents appreciated the ability of LLMs to simplify the learning process and reduce the time spent on mastering technical concepts. R-23 explained: "I think with language models you don't have to spend that much time learning 'technical' things." This can be linked to the contextual understanding and knowledge retention capabilities of LLMs, which allow them to provide tailored guidance and support for users with varying levels of expertise.

4.6.5 Customization and Personalization. Finally, customization and personalization were highlighted as advantages by 8% of respondents. LLMs were praised for their ability to provide more digestible information and adapt responses to user preferences. R-94 described the flexibility of LLMs: "It gives more digested information, and we can 'mold' the information how we want (e.g., asking the language model to respond using short sentences, or to explain in detail certain topics, etc)." This aspect of LLMs can be attributed to their capacity for understanding context and user preferences, allowing them to generate more relevant and personalized responses [72].

In summary, our thematic analysis revealed several key aspects of LLMs that contribute to their perceived relative advantage in software engineering, including time efficiency, code quality, user experience, learning and skill development, and customization and personalization. These findings

Table 7. Data Structure of Social Influence of LLMs in Software Engineering

ID	Quote	1st-Order Concepts	2nd-Order Themes	Aggregate Dimensions	(%)
R-66	My colleagues don't influence on my decisions to use language models	No influence	Independent decision making	No Influence	29
R-24	Not at all.	No influence	Independent decision making	No Influence	29
R-58	Some are in favor, but it doesn't influence my opinion or way of using it	Others in favor, no influence on opinion or usage	Independent evaluation	No Influence	29
R-97	Very much, as we use it on a daily basis and it is encouraged.	Use on a daily basis, encouraged	Strong adoption	High Influence	26
R-79	We all use them, so we encourage each other to use them as well	Collective use, mutual encouragement	Collaboration and support	High Influence	26
R-27	Greatly, everyone I know is keen on exploring their possibilities.	Strong influence, exploring possibilities	Enthusiastic adoption	High Influence	26
R-45	Not much	Minimal influence	Low level of influence	Low Influence	24
R-99	Not really.	Minimal influence	Low level of influence	Low Influence	24
R-9	I make my own decisions . . . but I often seek input and feedback from my colleagues and peers . . .	Independent decision, input and feedback from peers	Valuing peer opinions	Moderate Influence	21
R-82	There's some influence but I don't think it's a big deal because everyone uses it out of curiosity	Some influence, curiosity-driven adoption	Curiosity and exploration	Moderate Influence	21

align with DOI, suggesting that the adoption of LLMs in software engineering can be facilitated by their ability to provide clear benefits over existing methods [85]. Moreover, our results highlight the potential of LLMs to revolutionize the field of software engineering by streamlining processes, enhancing user experience, and fostering continuous learning and improvement.

4.7 Social Influence of LLMs in Software Engineering

The adoption of LLMs in software engineering is influenced by various factors. One key factor is the social influence from peers and colleagues, as suggested by SCT [9]. The current analysis aims to provide a rationale on how the data explains the "social influence" in relation to the adoption of LLMs in software engineering and how it links to the theoretical framework of SCT. Our analysis (Table 7) revealed four aggregate dimensions representing the range of social influences in the adoption of LLMs: No Influence, Low Influence, Moderate Influence, and High Influence.

4.7.1 No Influence. Our analysis revealed that 29% of the respondents reported no influence from their colleagues or peers on their decision to use LLMs (e.g., R-66, R-24, R-58). This finding suggests that a considerable proportion of software engineers make independent decisions about whether to adopt LLMs. This aligns with the literature on individual agency and self-efficacy in SCT [10]. These software engineers may rely on their own evaluation of the technology and personal preferences rather than the opinions or experiences of their colleagues.

4.7.2 Low Influence. Low influence was reported by 24% of the respondents (e.g., R-45, R-99). This indicates that some software engineers may be slightly influenced by their peers but ultimately retain a high degree of autonomy in their decision making. This finding suggests that while social influence may play a role in the adoption of LLMs, individual factors, such as personal interest and perceived utility, may also significantly contribute to the decision-making process [5].

4.7.3 Moderate Influence. Our analysis revealed that 21% of the respondents reported moderate influence from their peers and colleagues (e.g., R-9, R-82). This suggests that a significant proportion of software engineers value the input and feedback of their peers when deciding whether to adopt LLMs. This finding is consistent with SCT, which emphasizes the role of observational learning and vicarious experiences in shaping individual behavior [9]. In the context of LLM adoption, this moderate influence may result from a combination of individual factors and the experiences of colleagues.

4.7.4 High Influence. Finally, 26% of the respondents reported high influence from their peers and colleagues (e.g., R-97, R-79, R-27). This finding suggests that a considerable proportion of software engineers are strongly influenced by the collective use and enthusiasm for LLMs within their professional circles. This result aligns with SCT's focus on the reciprocal interactions between individuals and their social environment, as well as the literature on technology adoption in organizations [85]. In this case, the high level of influence may stem from the perceived benefits of LLMs, collaboration, and shared enthusiasm for exploring the technology's possibilities.

In summary, our analysis revealed a diverse range of social influence levels in the adoption of LLMs in software engineering. While some software engineers reported no influence from their colleagues or peers, others indicated low, moderate, or high levels of influence. These findings highlight the complex interplay between individual factors, such as self-efficacy and personal interest, and social influences, as posited by SCT. The understanding of these various levels of social influence can inform future research on the adoption of LLMs and other emerging technologies in software engineering, as well as organizational strategies for encouraging their appropriate use.

4.8 Self-Efficacy of LLMs in Software Engineering

Self-efficacy, an integral construct within SCT, refers to an individual's belief in their ability to perform specific tasks and achieve desired outcomes [10]. In the context of software engineering, the self-efficacy of developers using LLMs can impact their adoption and effective utilization. Based on our thematic analysis, we identified three aggregate dimensions that contribute to understanding self-efficacy in relation to LLMs adoption: (1) Importance of being seen as cutting-edge, (2) Focus on practicality and efficiency, and (3) Low importance of being seen as cutting-edge. These dimensions will be discussed in detail in the following, highlighting their role in shaping developers' self-efficacy and adoption behavior of LLMs in software engineering. The data structure can be found in Table 8.

4.8.1 Importance of Being Seen as Cutting-Edge. As shown in Table 8, 57% of respondents emphasized the importance of being seen as someone who uses cutting-edge technology in their work. This perception aligns with the notion of self-efficacy, as developers who consider themselves proficient in the latest technologies tend to have higher confidence in their capabilities [24]. For example, R-2 expressed that being up-to-date with cutting-edge technology is crucial in their line of work, and R-30 mentioned the fear of being replaced by someone perceived as more proficient in cutting-edge technology. This focus on staying current with technological advancements may encourage developers to adopt LLMs to showcase their expertise and maintain a competitive edge in the industry [110].

Table 8. Data Structure of Self-Efficacy of LLMs in Software Engineering

ID	Quote	1st-Order Concepts	2nd-Order Themes	Aggregate Dimensions	(%)
R-2	Being up-to-date with cutting-edge technology is crucial in my line of work.	Importance of cutting-edge	Emphasizing cutting-edge importance	Importance of being seen as cutting-edge	57
R-30	Very important. I don't want to be replaced with someone who's seen like that while I don't.	Fear of being replaced	Emphasizing cutting-edge importance	Importance of being seen as cutting-edge	57
R-57	It's important to show I'm capable of adapting.	Capability of adapting	Emphasizing cutting-edge importance	Importance of being seen as cutting-edge	57
R-77	I think is really important to be always up to date with new technologies	Importance of being up-to-date	Emphasizing cutting-edge importance	Importance of being seen as cutting-edge	57
R-10	it is important for me because it helps me to stay ahead of the curve and deliver innovative solutions that meet evolving customer needs.	Staying ahead, innovative solutions	Focus on practicality and efficiency	Focus on practicality and efficiency	35
R-19	In my work as a software developer, it is important for me to keep up to date with the latest trends and technologies in software engineering, including language models . . .	Meeting clients' needs, appropriate tech	Focus on practicality and efficiency	Focus on practicality and efficiency	35
R-47	Not much, my work is mostly based on the technologies others decide to use.	Using decided technologies	Focus on practicality and efficiency	Focus on practicality and efficiency	35
R-100	A little bit, but cutting-edge technology does not always mean higher efficiency. And efficiency is key.	Efficiency, not always cutting-edge	Focus on practicality and efficiency	Focus on practicality and efficiency	35
R-21	Not at all. I'm looking for stability, not implementing the next bleeding-edge thing in my workflow.	Stability, rejecting cutting-edge tech	Low importance of being seen as cutting-edge	Low importance of being seen as cutting-edge	8
R-66	I don't find it important to be seen as someone who uses cutting-edge technology in my work. My focus is on using the most effective tools for the task at hand.	Low importance of cutting-edge, effective tools	Low importance of being seen as cutting-edge	Low importance of being seen as cutting-edge	8

4.8.2 Focus on Practicality and Efficiency. Our analysis revealed that 35% of respondents highlighted the importance of practicality and efficiency in their work, demonstrating a preference for using technologies that effectively solve problems or meet client needs rather than simply being cutting-edge. This focus reflects a more task-oriented self-efficacy, where developers concentrate on finding the most appropriate tools for the job. R-10, for instance, emphasized the importance of staying ahead of the curve and delivering innovative solutions to meet evolving customer needs, whereas R-19 and R-100 mentioned the balance between adopting cutting-edge technologies and ensuring efficiency in their work. This suggests that developers with a focus on practicality and efficiency will adopt LLMs only when they perceive these models to provide tangible benefits to their work.

4.8.3 Low Importance of Being Seen as Cutting-Edge. A smaller group of respondents (8%) assigned low importance to being seen as using cutting-edge technology in their work. These

developers prioritize stability, effectiveness, or other factors over adopting the latest technologies, which may influence their self-efficacy in terms of LLM adoption [104]. For instance, R-21 expressed a preference for stability over implementing bleeding-edge technology, and R-66 stated that their focus is on using the most effective tools for the task at hand, regardless of whether they are cutting-edge. This finding suggests that developers with a low emphasis on cutting-edge technology may be less inclined to adopt LLMs unless they demonstrate clear benefits over existing tools.

In conclusion, our thematic analysis of self-efficacy in relation to LLM adoption in software engineering has identified three aggregate dimensions that provide valuable insights into developers' perceptions and behaviors. These dimensions suggest that the importance assigned to cutting-edge technology, along with the focus on practicality and efficiency, plays a critical role in shaping developers' self-efficacy and their likelihood of adopting LLMs.

4.9 Environmental Factors of LLMs in Software Engineering

The adoption of LLMs in software engineering is influenced by various environmental factors that shape organizational behaviors and decision-making processes. Drawing on the SCT framework [9], this study investigates how environmental factors affect the extent to which organizations are supportive of adopting LLMs as a standard technology. The following sections present the findings of our thematic analysis, which revealed five aggregate dimensions related to environmental factors: Supportive Attitude, Neutral Stance, Conditional Support, Limited Support, and Lack of Support. Each dimension is discussed in detail with references to the data presented in Table 9.

4.9.1 Supportive Attitude. The most prevalent aggregate dimension in our data, Supportive Attitude, captures the positive and proactive stance of organizations in promoting LLM adoption (42% frequency). This dimension encompasses strong organizational encouragement and investment in the technology to facilitate its integration into software engineering practices [85]. Respondent R-5, for instance, highlights the active promotion of LLMs by their organization, stating that they "pay for it and encourage us to use it." Similarly, R-45 reports a "very supportive" attitude, illustrating the extent to which some organizations prioritize the adoption of LLMs.

4.9.2 Neutral Stance. The Neutral Stance dimension reflects organizations that neither actively support nor oppose the adoption of LLMs (20% frequency). This stance may be attributed to the lack of awareness or knowledge about LLMs, or a wait-and-see approach to gauge the potential benefits and drawbacks of the technology [54]. Respondent R-84 describes their organization's position as "neutral, up to the employee," implying that the decision to use LLMs is left to individual discretion rather than being guided by organizational policy.

4.9.3 Conditional Support. Some organizations in our sample exhibit a Conditional Support dimension (19% frequency), characterized by their willingness to adopt LLMs provided certain criteria are met, such as the technology demonstrating clear benefits or aligning with specific organizational objectives [85]. In this context, R-26 notes that their organization supports LLM adoption "if it brings more advantages to the team and the way we work, and make things faster."

4.9.4 Limited Support. The Limited Support dimension (12% frequency) represents organizations that only support the use of LLMs in specific contexts or for certain tasks. This selective approach may stem from concerns related to security, privacy, or ethical considerations. For example, R-64 explains that their organization "opposes them when working on new features but for debugging, they can be quite helpful."

Table 9. Data Structure of Environmental Factors of LLMs in Software Engineering

ID	Quote	1st-Order Concepts	2nd-Order Themes	Aggregate Dimensions	(%)
R-5	A lot, they pay for it and encourage us to use it	Encouragement, payment	Active promotion	Supportive Attitude	42
R-45	Very supportive	Strong support	Highly supportive	Supportive Attitude	42
R-84	I think my organization is neutral, is up to the employee	Neutral, employee choice	Lack of strong opinion	Neutral Stance	20
R-26	If it brings more advantages to the team and the way we work, and make things faster, they are supportive of adopting language models.	Advantages, efficiency, conditional support	Support with conditions	Conditional Support	19
R-21	Not at all. We are anti-AI art, and also anti-AI when it comes to code. It failed to comply with our requirements anyway in testing.	Not supportive, anti-AI	Opposition to AI technology	Lack of Support	15
R-74	Not much, since it might break copyright and also poses some security concerns about proprietary intellectual property	Copyright concerns, security concerns, limited support	Concerns leading to lack of support	Lack of Support	15
R-92	I don't know, they don't oppose it though	Uncertainty, no opposition	Unsure	Uncertainty	14
R-61	My organization is somewhat supportive but they are not sure	Partial support, uncertainty	Ambivalence	Uncertainty	14
R-79	They're supportive of using it as a support tool, not as a main tool	Support, limited context	Limited application	Limited Support	12
R-64	They oppose them when working on new features but for debugging, they can be quite helpful.	Opposition, debugging, specific context	Support in specific contexts	Limited Support	12

4.9.5 Lack of Support. Finally, the Lack of Support dimension (15% frequency) captures organizations that actively oppose or discourage the use of LLMs in software engineering. This stance may be driven by various factors, such as ethical concerns, fear of job displacement, or skepticism about the technology's effectiveness. Respondent R-74 reveals that their organization offers limited support for LLMs, mainly due to "copyright and security concerns about proprietary intellectual property."

In summary, our thematic analysis of environmental factors affecting LLM adoption in software engineering has identified five aggregate dimensions, ranging from strong support to active opposition. These dimensions provide valuable insights into the diverse organizational attitudes and contexts that shape the integration of LLMs into software engineering practices, contributing to a deeper understanding of the role of environmental factors in the SCT framework.

4.10 Key Insights of the Qualitative Study

Our findings demonstrate the potential benefits of LLMs in software engineering, highlighting their impact on automating repetitive tasks, enhancing problem-solving abilities, facilitating learning and understanding, improving code quality, assisting in debugging and optimization, and increasing overall efficiency. However, concerns about the limitations of LLMs, such as their lack of knowledge about internal APIs and the need for human oversight, emphasize the importance of human expertise in utilizing these tools effectively.

The perceived ease of use of LLMs in software engineering is influenced by factors such as integration with existing tools and workflows, accessibility and comprehensibility of documentation, customizability and adaptability, and support from the developer community. These factors are crucial in facilitating the adoption of LLMs and their seamless integration into the software engineering domain.

The behavioral intention to adopt LLMs in software engineering is shaped by the perceived usefulness, perceived ease of use, social influence, and facilitating conditions. These factors collectively contribute to creating an environment conducive to the successful integration of LLMs into software engineering practices.

Regarding DOI, the compatibility of LLMs in software engineering is influenced by dimensions such as improved efficiency, assistance and support, similarity to current practices, and adaptation and learning. However, concerns about dependency and skill degradation, privacy, security and data protection, job displacement, and labor market implications, and accuracy, reliability, and explainability underline the complexity of LLM adoption in this domain.

Our findings also reveal the relative advantage of LLMs over traditional methods in software engineering. Factors such as time efficiency, code quality, user experience, learning and skill development, and customization and personalization contribute to the perceived benefits of LLMs in this domain.

From an SCT perspective, the influence of peers and colleagues on LLM adoption in software engineering varies from no influence to high influence. Self-efficacy is influenced by factors such as the importance of being seen as cutting-edge, a focus on practicality and efficiency, and the low importance of being seen as cutting-edge. Environmental factors, such as supportive organizational culture, uncertainty, security concerns, neutral organizational culture, resistance to change, and limited or marginal support, also play a role in LLM adoption.

In conclusion, the adoption of LLMs in software engineering is a multi-faceted phenomenon influenced by a variety of factors and theoretical perspectives. Our study provides valuable insights into the key dimensions and concerns that shape the integration of LLMs in the software engineering domain, paving the way for future research and practice in this area.

5 THE HUMAN-AI COLLABORATION AND ADAPTATION FRAMEWORK

In this section, we introduce the **Human-AI Collaboration and Adaptation Framework (HACAF)**, an innovative theoretical model designed to understand and predict the adoption of Generative AI tools in software engineering. HACAF derives its components from several established theories including TAM, DOI, SCT, the **Unified Theory of Acceptance and Use of Technology (UTAUT)**, and the personal innovativeness construct.

While the original TAM, DOI, and SCT theories provide robust theoretical foundations for understanding technology acceptance and adoption, our qualitative investigation indicated the necessity for a more nuanced model. HACAF is, therefore, not merely an amalgamation of these theories but also an evolution, as it incorporates additional facets revealed in our research.

The inclusion of constructs from UTAUT addresses the need for a greater focus on social influence and facilitating conditions, elements that emerged as significantly influential in our qualitative findings. The additional integration of the personal innovativeness construct into HACAF is motivated by the observed variability in adoption behaviors among software engineers, even within the same contextual environment, implying a role for individual differences in innovative tendencies.

By merging these four main components into HACAF, we not only leverage the collective strengths of these prominent theories but also account for the additional complexities of technology adoption that surfaced in our qualitative investigation. Consequently, HACAF represents

a tailored approach that reflects the multi-faceted nature of LLM adoption in software engineering. This comprehensive framework aims to provide a deeper understanding of the complex dynamics involved in the adoption of LLMs and act as a guide for future research and practice in this rapidly evolving domain.

Perceptions about the technology is a cornerstone of HACAF, rooted in TAM. This construct denotes a software engineer's evaluation of the usefulness, ease of use, and relative advantage of LLMs. The qualitative data substantiates this construct by revealing that software engineers assess LLMs based on their potential to streamline coding processes, enhance code quality, and expedite project timelines. Furthermore, the focus on practicality and efficiency found in our study underscores the importance of this construct in influencing the adoption of LLMs.

Compatibility factors, informed by DOI, illustrate the degree to which LLMs align with the existing values, experiences, and needs of potential adopters. Our qualitative investigation underpins this construct by highlighting the importance of the fit of LLMs within current software development workflows. Developers who perceive a high degree of fit, irrespective of the technology's cutting-edge nature, are more likely to adopt LLMs, reinforcing the relevance of this construct.

Social factors, drawing on UTAUT and the concept of computer self-efficacy, emphasize the influence of the social environment and an individual's belief in their abilities to use LLMs. The investigation's findings lend weight to this construct by indicating that the importance of being seen as cutting-edge and the developer's self-efficacy could drive LLM adoption. The role of peer approval and the belief in one's competence to master LLMs surfaced as significant influencers, further establishing this construct's salience in the HACAF model.

Personal and environmental factors bring into play the role of personal innovativeness and organizational support. Personal innovativeness represents an individual's predisposition toward new technologies, and organizational support captures the perceived facilitating conditions within an organization for the use of LLMs. Both elements emerged as significant in our investigation. Our data evidenced how individual readiness to experiment with LLMs and the organization's stance toward LLMs, ranging from active support to outright opposition, directly influence the likelihood of LLM adoption.

In conclusion, the HACAF model, informed and justified by our qualitative investigation and underpinned by established theoretical constructs, offers a nuanced understanding of the interplay of individual and organizational factors influencing the adoption of LLMs in software engineering. By grounding this framework in both empirical evidence and theory, we aim to provide a solid foundation for further empirical scrutiny and contribute to the understanding of AI technology adoption dynamics.

5.1 Theoretical Model and Hypotheses

The previous section's theoretical foundation, bolstered by the qualitative investigation, serves as the basis for operationalizing HACAF. We now translate these theoretical constructs into four main determinants of the intention to use LLMs: perceptions about the technology, compatibility factors, social factors, and personal and environmental factors, and formulate hypotheses based on these constructs, graphically represented in Figure 1.

Perceptions about the technology encapsulate the perceived usefulness, ease of use, and relative advantage of LLMs. As our qualitative data revealed, LLMs' perceived usefulness and relative advantage, such as streamlining coding processes and enhancing code quality, have a significant bearing on their adoption. The ease of use was another critical factor, as intuitive and user-friendly LLMs are more likely to be adopted by software engineers. Thus, drawing upon TAM, we propose the following hypotheses.

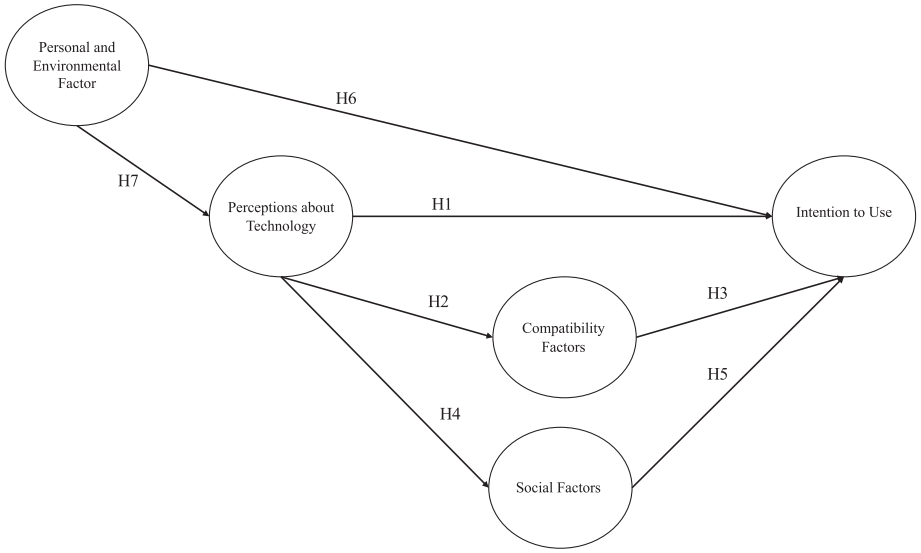

Fig. 1. The HACAF.

Hypotheses. *H1: Positive perceptions about the technology (PT), encompassing perceived usefulness, ease of use, and relative advantage, will increase the Intention to Use (IU) LLMs in a software engineering context.*

Compatibility factors address the extent to which LLMs align with the existing values, experiences, and needs of potential adopters. As the qualitative investigation underlines, LLMs' compatibility with current software development workflows significantly influences adoption decisions. Software engineers are more likely to adopt LLMs when they perceive a high degree of fit with their work practices. Following this and DOI, we hypothesize the following.

Hypotheses. *H2: Positive perceptions about the technology (PT) will enhance the Compatibility Factors (CF). H3: Enhanced compatibility factors (CF) will in turn increase the Intention to Use (IU) LLMs.*

Social factors, which include social influence and self-efficacy, play an important role in adoption decisions. Our qualitative study emphasized that peer approval and an individual's belief in their ability to use LLMs could significantly influence the intention to use these technologies. Based on this and UTAUT and the concept of computer self-efficacy, we posit the following.

Hypotheses. *H4: Positive perceptions about the technology (PT) will increase Social Factors (SF). H5: Increased social factors (SF) will enhance the Intention to Use (IU) LLMs.*

Finally, *personal and environmental factors*, including personal innovativeness and organizational support, contribute to the complexity of the model. As underscored by our investigation, an individual's willingness to experiment with LLMs and the perceived supportiveness of the organization can be decisive for LLM adoption. Thus, we hypothesize the following.

Hypotheses. *H6: Personal and Environmental Factors (PEF), specifically personal innovativeness and organizational support, will moderate the relationship between Perceptions about the technology and IU LLMs, strengthening the positive effect of Perceptions about the technology on Intention to Use LLMs.*

6 THEORY VALIDATION

In the subsequent phase of our research, we transitioned from the qualitative insights garnered in the initial study to a quantitative validation. The qualitative findings from our initial study played a foundational role in shaping the subsequent phases of our research. Also here, we used the SIGSOFT Empirical Standard for Questionnaire Surveys and Multi-Methodology and Mixed Methods Research [79].

Partial Least Squares–Structural Equation Modeling. This method was chosen based on its ability to validate complex models, especially when the research is in an exploratory stage, as is the case with our study on the adoption dynamics of Generative AI tools in software engineering.

Scale Development. The themes and patterns identified in our qualitative findings were instrumental in developing the scales for our quantitative study. These scales were designed not only to encapsulate the core of our qualitative insights but also to render them into measurable metrics. In line with established methodological guidelines [91], we adapted existing scales to ensure robustness and relevance. A detailed breakdown of these scales can be found in Table 20 in Appendix B.

Survey Data Collection. The design of our survey was directly informed by the conclusions drawn from our qualitative exploration. Questions were framed to test the hypotheses that emerged from the qualitative study, ensuring a coherent flow between the two research phases.

Participant Demographics. The demographics were chosen based on the insights from the qualitative study to ensure that the quantitative study sample was representative of the broader population of software engineers who might be impacted by the adoption of Generative AI tools.

Evaluation of the Measurement Model. The constructs used in the measurement model were derived from the themes of the qualitative study. This ensured that the quantitative analysis was grounded in the real-world experiences and perceptions of our initial study participants.

Evaluation of the Structural Model. The relationships tested in the structural model were informed by the patterns and relationships identified in the qualitative study. This ensured that our quantitative validation was directly aligned with the insights from our initial exploration.

6.1 Partial Least Squares–Structural Equation Modeling

PLS-SEM is a multi-faceted statistical examination designed to substantiate latent and unseen variables (or constructs) through multiple observable indicators. This method is particularly useful for theory development studies and is increasingly adopted in empirical software engineering [91]. PLS-SEM can address multiple interconnected research queries in one extensive analysis, making it a popular choice across other fields such as Management, Information Systems Research, and Organizational Behavior. As Gefen et al. [37] suggest, SEM is commonly employed to authenticate tools and verify connections between constructs. The subsequent evaluation and analysis of the PLS-SEM model adhere to the latest guidelines and recommendations for research in software engineering by Russo and Stol [91].

6.1.1 Scale Development. The survey was crafted with the assistance of supplementary theory. We structured our survey by adapting instruments from previous research. All items utilized to define each construct and the references used to shape the questions are summarized in Table 20 in Appendix B. Each construct was assessed through uni-dimensional items on a 7-point Likert scale indicating levels of agreement.

Initially, a pre-test was conducted with three potential respondents (software engineers) to assess the survey's usability, reasoning, and phrasing. The usability received positive feedback, and some minor issues with the reasoning and phrasing were identified and subsequently addressed.

6.1.2 Survey Data Collection. The minimum sample size was determined by conducting an *a priori* power analysis with G*Power. With an effect size of 15%, significance level at 5%, and a power of 95%, the smallest size required for seven predictors is 153.

A cluster sampling strategy was utilized for data collection, facilitated through the academic data collection platform, Prolific. The survey was delivered via Qualtrics, randomizing the order of questions within their blocks to reduce response bias.

A multi-phase screening process was implemented to ensure the integrity of the collected data. Data collection was carried out between April and June 2023.

Pre-Screening. During the initial selection phase, participants were chosen based on specific self-reported criteria. These included proficiency in *Computer Programming*, employment in the *Software* industry on a *Full-time* basis, a non-student status, and an *Approval rate* of 100%. On Prolific, the approval rate represents the percentage of a participant's submissions that have been approved by researchers, indicating their reliability and consistency in providing quality responses. To emphasize our target demographic, our survey was titled "[SOFTWARE ENGINEERS ONLY] Human-AI Collaboration and Adaptation." This title was chosen to clearly communicate to professionals who met the pre-screening criteria but were not actively working as software engineers. In total, 831 potential participants met these criteria.

Competence Screening. To ensure the accuracy of our pool, participants were asked to complete a questionnaire containing three competency-based questions about software design and programming. Additionally, they conveyed their familiarity with, and usage of, Generative AI tools, at least to a certain extent. This helped confirm the reliability of their self-reported skills. This process resulted in a reduced pool of 606 participants.

Quality and Competence Screening. To ensure the accuracy of our pool, we added one single-item screening question in our survey from Danilova et al. [28], asking "Which of these websites do you most frequently use as aid when programming?" with 'Stack Overflow' as the correct question. Additionally, we added three random attention checks to further ensure data quality. A total of 220 completed questionnaires were received, but 36 were discarded due to failure on at least one attention check. This left us with 184 valid and complete responses, surpassing the minimum sample size.

6.1.3 Participant Demographics. Our survey encompassed a diverse set of 184 respondents, comprising 80% males, 18% females, 1% non-binary individuals, and 1% who preferred not to disclose their gender. The respondents were drawn from a broad geographic pool spanning 27 unique countries, with the most populous responses originating from the UK (24%), South Africa (13%), Poland (11%), Germany (11%), and the United States of America (7%).

In terms of work tenure, the median experience among participants was 3 years. The majority, 125 respondents, were relatively early in their careers, with 1 to 5 years of experience. Forty participants reported a more substantial work experience ranging between 6 and 15 years. An additional 14 participants had an extensive work experience of 16 to 30 years, and a handful of respondents, 5 in total, possessed more than 30 years of experience.

Our sample prominently featured individuals from the software development sector, making up 66% of all respondents. Additionally, 12% of respondents held data analysis, engineering, or science roles. A smaller segment, 8%, held leadership roles such as Team Leads or CIOs. The remaining respondents included Tester/QA Engineers (6%), DevOps/Infrastructure Engineers (3%), Architects (2%), UX/UI Designers (2%), and other roles (2%).

Table 10. HTMT of the model

	CF	IU	PEF	PT
Compatibility Factors (CF)				
Intention to Use (IU)	0.756			
Personal and Environmental Factors (PEF)	0.372	0.272		
Perceptions about the Technology (PT)	0.848	0.633	0.338	
Social Factors (SF)	0.403	0.371	0.441	0.437

Table 11. Internal Consistency Reliability

	Cronbach's Alpha	rho_a	rho_c	AVE
Compatibility Factors (CF)	0.856	0.866	0.902	0.699
Intention to Use (IU)	0.939	0.941	0.961	0.891
Personal and Environmental Factors (PEF)	0.876	0.905	0.914	0.727
Perceptions about the Technology (PT)	0.948	0.950	0.956	0.685
Social Factors (SF)	0.875	0.906	0.940	0.887

6.2 Evaluation of the Measurement Model

To ensure the validity and reliability of our structural model, it is paramount to evaluate the reliability of the latent variables. Consequently, we analyze the discriminant validity, internal consistency reliability, and convergent validity initially.

6.2.1 Discriminant Validity. In this context, discriminant validity refers to the distinctness or uniqueness of one latent variable compared to another. This serves as an essential parameter for determining whether two constructs are essentially the same and represent varying facets of knowledge. For its evaluation, we utilized the **Heterotrait-Monotrait ratio of correlations (HTMT)**, which is recognized for its superior performance over other tests like the Fornell-Larcker criterion. The HTMT values should ideally be below 0.90.

Table 10 reveals that all coefficients fall beneath the pre-defined threshold, which suggests that every construct in the model represents a distinct phenomenon.

6.2.2 Internal Consistency Reliability. This test seeks to confirm that the items are gauging the latent variables in a consistent and reliable manner. As such, we refer to the Cronbach's alpha, rho_a, and rho_c values showcased in Table 11, all of which should exceed 0.60 [70]. We can conclude that our tests meet the reliability criterea.

6.2.3 Convergent Validity. The final validity assessment examines the extent of correlations between various items and their corresponding construct. It is noteworthy that our latent variables are reflectively measured (Mode A).[2] As a result, these indicators should demonstrate a substantial variance proportion by converging on their latent variables. Two tests were employed to verify this assumption.

The first test involves the average variance extracted (AVE), which should register a value exceeding 0.5 [46]. The second test involves ensuring that the outer loadings of each measurement model for the latent variable account for at least 50% variance. This is tested by assessing the indicator's reliability, which should exceed the square root of 50% (i.e., 0.7).

Table 12 encapsulates the results of the indicator's reliability using cross loadings. The items that did not contribute significantly to the variance and were subsequently excluded from our

[2]For a comprehensive comparison between reflective and formative measures, see the work of Russo and Stol [91].

Table 12. Cross Loadings (Full List of Items Is Presented in Table 20 in Appendix B)

Item	CF	IU	PEF	PT	SF
CF2	**0.896**	0.626	0.293	0.686	0.333
CF3	**0.856**	0.657	0.352	0.666	0.305
CF5	**0.775**	0.482	0.248	0.547	0.250
CF6	**0.811**	0.505	0.236	0.654	0.289
IU1	0.662	**0.935**	0.238	0.560	0.293
IU2	0.602	**0.945**	0.235	0.558	0.337
IU3	0.671	**0.951**	0.267	0.577	0.330
PEF4	0.293	0.198	**0.811**	0.276	0.277
PEF5	0.156	0.130	**0.838**	0.214	0.250
PEF6	0.362	0.316	**0.892**	0.300	0.394
PEF7	0.298	0.200	**0.867**	0.258	0.391
PT1	0.658	0.581	0.251	**0.841**	0.340
PT11	0.641	0.521	0.178	**0.855**	0.318
PT12	0.490	0.415	0.362	**0.703**	0.350
PT13	0.639	0.491	0.295	**0.856**	0.327
PT14	0.612	0.460	0.267	**0.851**	0.330
PT2	0.663	0.545	0.281	**0.827**	0.378
PT3	0.676	0.503	0.262	**0.854**	0.344
PT4	0.655	0.511	0.271	**0.874**	0.383
PT6	0.669	0.550	0.234	**0.866**	0.323
PT7	0.622	0.353	0.191	**0.727**	0.243
SF1	0.295	0.291	0.388	0.325	**0.929**
SF2	0.365	0.342	0.359	0.427	**0.955**

model during the analysis phase (a complete list of excluded items can be found in Table 20 in Appendix B). Consequently, an improvement in the AVE was noted, thereby reinforcing the model's robustness.

6.3 Evaluation of the Structural Model

After ensuring the reliability of all of our constructs through our measurement model assessment, we can now shift our focus toward evaluating the structural model, graphically represented in Figure 2. This evaluation is pivotal in discussing the predictive power of our model and validating our research hypotheses.

6.3.1 Collinearity Analysis. Initially, we analyze the correlation between the exogenous variable (Personal and environmental factors) and other endogenous variables. These should be independent to prevent any potential bias in the path estimations. The variance inflation factor test, which detects multicollinearity (i.e., an extreme degree of collinearity), should have a value under 5 [63]. Our variance inflation factor values are below this threshold, ranging from 4.710 (for the IU_3 item) to 2.034 (PEF_4). Consequently, we deduce that our model does not suffer from multi-collinearity issues.

6.3.2 Path Relations: Significance and Relevance. Path coefficients represent the hypothesized relationships between latent variables. They are standardized, meaning their values can range from −1 to +1. PLS-SEM does not require distributional assumptions, which means that we cannot

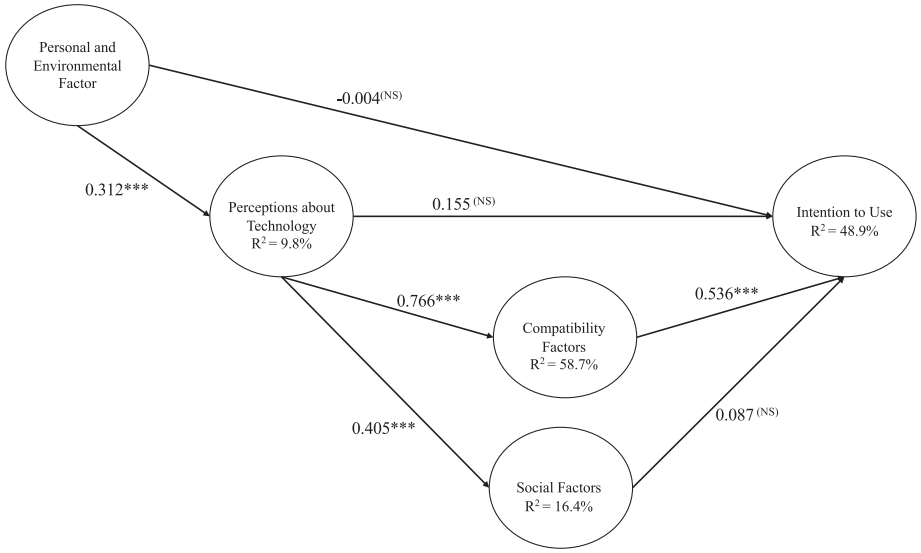

Fig. 2. HACAF's structural model with R^2 and path coefficients (***p< 0.001, (NS) p > 0.05).

Table 13. Path Coefficients, Bootstrap Estimates, Standard Deviation, T Statistics, and p-Values

Hypotheses	Coefficient	Bootstrap Mean	St. Dev.	T	p
H1: PT → IU	0.155	0.150	0.122	1.271	0.204
H2: PT → CF	0.766	0.766	0.047	16.405	0.000
H3: CF → IU	0.536	0.537	0.090	5.936	0.000
H4: PT → SF	0.405	0.408	0.076	5.346	0.000
H5: SF → IU	0.087	0.090	0.064	1.373	0.170
H6: PEF → IU	−0.004	−0.004	0.056	0.064	0.949
H7: PEF → PT	0.313	0.317	0.072	4.313	0.000

use parametric tests to assess significance. As a workaround, we use a two-tailed bootstrapping method that incorporates 5,000 sub-samples with replacement.

Details of the bootstrapping results are presented in Table 13, which includes the bootstrapping coefficients, mean, standard deviation, T statistics, and p-values corresponding to each of our seven hypotheses.

Our analysis reveals that a majority of our hypotheses are statistically significant. Specifically, four relationships have p-values less than 0.05, and their T statistics surpass 1.96, indicative of a 5% significance level as noted by Hair et al. [46].

6.3.3 Evaluation of Determination Coefficients. After affirmatively determining the significance of the majority of our hypotheses, we now proceed to the final phase of our study, which centers on the predictive power of the endogenous constructs. This is shown in Table 14. The predictive capacity is quantified through the variance explained (R^2) by the endogenous constructs. R^2 signifies the ratio of the variance in the dependent variable that can be predicted from the independent variables. As R^2 increases with the number of predictors (more predictors equate to a higher R^2), it is prudent to consider the adjusted R^2 as well, which takes into account the quantity of predictors in the model. The values for these metrics lie between 0 and 1. Determining a standard for R^2

Table 14. Coefficients of Determination

Construct	R^2	R^2 Adjusted
Compatibility Factors	0.587	0.585
Intention to Use	0.489	0.477
Personal and Environmental Factors	0.098	0.093
Social Factors	0.164	0.159

Table 15. Constructs Prediction Summary

Construct	RMSE	MAE	Q^2_{predict}
Compatibility Factors	0.089	0.984	0.752
Intention to Use	0.046	1.012	0.761
Perceptions about the Technology	0.076	0.998	0.733
Social Factors	0.078	0.971	0.755

Table 16. Effect Sizes (f^2)

Construct	CF	IU	PEF	PT	SF
Compatibility Factors		0.225			
Intention to Use					
Personal and Environmental Factors	0.000		0.108		
Perceptions about the Technology	1.426				0.196
Social Factors	0.018	0.011			

can be challenging, as its significance heavily relies on the topic at hand [46], but it is generally accepted that it should exceed 0.19 [17].

6.3.4 Evaluating Predictive Efficacy. Given that our study's primary objective is to ascertain predictions rather than causality, we undertook a predictive results evaluation employing PLSpredict [95]. This approach tests if a model (developed using a training sample) can predict the outcomes from a test sample. Our sample was segmented into 10 parts, and 10 repetitions were utilized to derive the PLSpredict statistics. The results were interpreted based on the guidelines proposed by Shmueli et al. [96]. Table 15 portrays the latent variables' predictive accuracy. Notably, all variables display a strongly positive Q^2_{predict} indicating robust performance of the model.

6.3.5 Assessing Predictive Consistency. Our final examination focuses on the predictive consistency of our model, for which we scrutinize the effect sizes (f^2) as illustrated in Table 16. This assessment involves understanding the impacts of various relationships within the model. The threshold values for effect sizes are set at 0.02, 0.15, and 0.35, corresponding to small, medium, and large effects, respectively [23]. Here, the relationship with the strongest effect are compatibility factors with intention to use LLMs.[3]

6.4 Determining Key Factors for Adoption: An Analysis Using the Importance-Performance Map

This focused study specifically explores the elements influencing the adoption and intended use of Generative AI tools in the field of software engineering. Using the **Importance-Performance Map Analysis (IPMA)** methodology, we have combined the analysis of both the importance and

[3]Although larger effect sizes are not inherently problematic, they can occasionally suggest a potential risk of overfitting. However, we have performed a comprehensive examination of this potential issue in Appendix B and concluded that overfitting is not present in our model.

Table 17. Constructs Performance Referred to Project Success

Construct	Construct Performance
CF	83.257
PEF	65.519
PT	85.138
SF	67.161

Table 18. Constructs Importance (Unstandardized Total Effects)

Construct	CF	IU	PEF	PT	SF
CF		0.646			
IU					
PEF	0.240	0.167		0.313	0.127
PT	0.767	0.540			0.405
SF		0.110			

performance dimensions derived from the PLS-SEM investigation [83]. This approach allows us to determine the extent to which various constructs contribute to the enhancement of the target construct—in this case, the Intention to Use (IU) and Adoption of Generative AI tools. It provides strategic insights by identifying which constructs are most significant and which ones demand improvements in performance.

Table 17 shows that all identified constructs, namely Compatibility Factors (CF), Personal and Environmental Factors (PEF), Perceptions about the Technology (PT), and Social Factors (SF), demonstrate robust performance, all exceeding 65%, with PT and CF even surpassing the 83% mark. This result is noteworthy, especially considering that established models such as TAM typically show a constructs' performance range between 50% and 70% [80].

The importance of individual constructs as shown in Table 18 is consistent with those of mature models, with values between 0.110 and 0.767. In particular, PT and CF emerge as the most significant constructs influencing IU. This finding emphasizes the role of perceptions about the technology and compatibility factors in the intention to use and the eventual adoption of Generative AI tools in software engineering.

Figure 3 provides a visualization of the interplay between the importance and performance of these constructs. For instance, a unit increase in the performance of PT (from, say, 85.138 to 86.138) would improve the IU by the total effect of PT on IU, which is 0.540. This suggests that if the goal is to increase IU and Adoption, emphasis should be placed on enhancing PT, given its high importance. Similarly, also CF play a crucial role to support AI adoption. However, constructs such as SF and PEF, despite their role, appear less critical to the intention to use and adoption of Generative AI tools.

7 DISCUSSION

In our investigation of Generative AI adoption in software engineering, we developed HACAF to dissect the interplay among perceptions about the technology, compatibility factors, social factors, and personal and environmental factors. We did not to restrict our study to a specific Generative AI model, such as ChatGPT-4 or ChatGPT3.5. This decision was driven by our aim to capture a holistic understanding of the adoption dynamics, recognizing that the landscape of Generative AI is rapidly evolving. Committing to a single model could have limited the generalizability and longevity of our

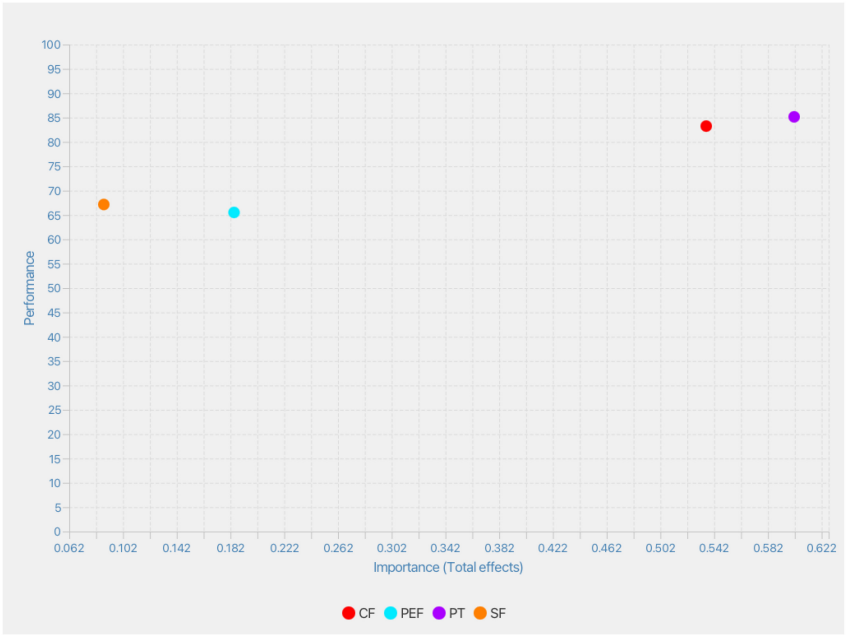

Fig. 3. IPMA of intention to use.

findings. By adopting a broader approach, we hypothesize that our framework may offer valuable insights, potentially remaining relevant as new iterations and variations of Generative AI models emerge in the field, although this supposition invites further empirical validation. The results revealed a complex landscape, with surprising deviations from traditional technology acceptance theories. Table 19 summarizes our key findings and implications.

Our qualitative study was foundational in the development of our framework, which stresses the impact of perceptions about the technology and compatibility factors in shaping the intention to use LLMs. However, the subsequent quantitative study upended the expectation that these perceptions directly influenced intention to use, thus contradicting the traditional TAM [29]. This suggests that when adopting LLMs, the compatibility with existing work processes significantly impacts perceptions about the technology.

The central role of compatibility factors aligns with previous findings [65] and reaffirms the essential need for technological alignment with current practices. Our study revealed that software engineers are more inclined to adopt LLMs when the technology fits seamlessly into their existing workflows, a finding in line with prior research [32].

In contrast to expectations, the quantitative investigation indicated that social factors did not significantly contribute to the intention to use LLMs. This deviation from previous studies [111] underlines the nuanced influences of social aspects and self-efficacy on the adoption process.

Personal and environmental factors, while influential in shaping perceptions about the technology, did not directly impact the intention to use. This observation supports the assertion by Agarwal and Prasad [2] that personal innovativeness might mold perceptions about a technology but does not necessarily guarantee adoption.

The insights derived from our exploration of LLM adoption using the HACAF theory both diverge from and extend beyond established theoretical frameworks like TAM, DOI, and SCT, shedding light on the complex dynamics at play. These findings carry both academic and practical

Table 19. Summary of Findings and Implications

Hypotheses	Findings	Implications
H1: Perceptions about the technology → Intention to Use	Not supported. We did not find a direct relationship between Perceptions about the technology and Intention to Use.	Perception of the technology alone does not instigate the adoption of a Generative AI tool.
H2: Perceptions about the technology → Compatibility Factors	Supported. This relationship is the strongest of the model with a path coefficient of 0.77 with a very large effect size (1.43).	Assimilating the capabilities of Generative AI tools and their potential integration into existing software development workflows is crucial.
H3: Compatibility Factors → Intention to Use	Supported. This relation is symmetrical to the H2 hypotheses (with a considerable path coefficient of 0.54 and a large effect size of 0.66), which mostly explains the intention to use of LLMs. The IPMA shows that Compatibility Factors are the most important element to improve the Intention to Use. Additionally, they explain, almost single handed, the adoption of AI tools with a very high R^2 close to 50%.	The adoption of Generative AI tools hinges largely on their successful integration with existing software development workflows.
H4: Perceptions about the technology → Social factors	Supported. We report a substantial path coefficient (0.40) with a medium effect size (0.20). The way Generative AI is perceived positively influences developer's self-efficacy.	Developers consider LLMs as vital facilitators in completing tasks more effectively.
H5: Social factors → Intention to Use	Not supported. The relationship between Social factors and Intention to Use is not significant.	Although Generative AI tools are deemed important enablers of developers' self-efficacy, this perception does not translate into adoption.
H6: Personal and environmental factors → Intention to Use	Not supported. This relationship is not significant.	Personal innovativeness and organizational support do not significantly influence developers' adoption of Generative AI tools.
H7: Personal and environmental factors → Perceptions about the technology	Supported. We found a significant relationship between Personal and environmental factors and Perceptions about the technology with a path coefficient of 0.31 and a small to medium effect size (0.11).	As expected, a developer's propensity to experiment with Generative AI tools, along with perceived organizational support, positively impact the perception of the technology.

implications, underlining the necessity for holistic strategies that consider individual perceptions and compatibility factors for promoting LLM adoption.

Our research places a clear emphasis on Compatibility Factors as primary drivers in the adoption process of Generative AI technologies. Compatibility factors, in essence, measure how well a new technology fits into an individual or organization's existing frameworkboth in terms of practical workflow and broader values. They are essential in determining the successful adoption of technologies, such as Generative AI. The importance of seamless integration within existing

workflows is the core driver toward AI adoption. Workflows refer to the defined sequence of tasks that a software engineering team performs regularly, such as coding, debugging, testing, and deployment. Notably, also hardware plays a crucial role in these processes [36]. If an LLM can integrate smoothly into these steps, such as by automating certain coding tasks or improving debugging efficiency, it is seen as highly compatible. Conversely, if the integration of LLM disrupts these established procedures or necessitates significant changes to existing operations, it might be deemed less compatible, and its adoption may face resistance. This idea extends beyond workflows to include the broader technological environment. If the LLM requires different operating systems or specific hardware that is not in use, or if it relies on knowledge or skills that the team does not possess, it can make the tool less compatible. Therefore, the technology compatibility and the alignment with existing skills are vital considerations. In other words, even if a tool holds significant potential utility, if an individual or an organization fails to understand how it fits within their existing framework (i.e., workflow, technical environment, or value system), its adoption becomes less likely. This insight brings into focus the critical role of compatibility in designing and promoting new technology. For successful integration within the software engineering industry, Generative AI tools should be designed with a deep understanding of the existing systems, practices, and values in mind.

Another noteworthy aspect of our findings is the non-significant relationship between Social Factors and Intention to Use, despite the positive influence of perceptions about the technology on Social Factors. This unexpected finding could be contextualized by considering the nascent stage of the Generative AI transformation within the industry. Despite the recognition of the potential benefits and enhancements to self-efficacy offered by LLMs, this does not necessarily catalyze immediate widespread adoption, suggesting that mere perceptions about a technology's potential advantages may not be sufficient to prompt its adoption.

Interestingly, we found that Personal and Environmental Factors did not directly lead to the Intention to Use, stressing the early stage of LLM adoption. In this initial phase, personal innovativeness and organizational support appear to exert limited influence on adoption, placing the emphasis squarely on Compatibility Factors. This suggests that as AI tools are more seamlessly integrated within existing workflows, their likelihood of adoption increases.

It is critical to acknowledge that we are currently in the nascent stages of the Generative AI transformation. As the utilization of these tools gains wider traction over time, we anticipate that the relationships within HACAF will be further corroborated. The current non-validation of all relationships within our framework model does not detract from its utility but offers initial insights into the factors shaping the adoption of Generative AI tools at this stage.

While our study primarily focused on the adoption dynamics of Generative AI tools in software engineering, it is worth noting that a minority of our informants did touch upon the ethical implications and potential risks associated with their use. Despite the limited number of informants addressing these concerns, the issues they raised are significant. LLMs, while powerful, can produce harmful outputs or inadvertently perpetuate biases present in the data they were trained on. Such outputs can have unintended consequences, especially when integrated into software products that reach a wide audience. Recent research, including that by Tsamados et al. [106], has emphasized the security vulnerabilities inherent to AI, especially with LLMs. Their findings suggest a need for a comprehensive understanding of these models' capabilities and vulnerabilities to ensure a balanced cost-benefit analysis. In the context of our study, it is crucial to recognize that while the compatibility of AI tools within existing development workflows is a significant driver for adoption, this should not overshadow the potential risks. Model providers, in their journey to promote adoption, should be proactive in communicating potential vulnerabilities and offering guidance on

best practices to mitigate risks. Given the rapid evolution of LLM technologies, periodic reviews and audits can ensure that models remain transparent and adhere to ethical standards. Platforms integrating LLMs should bolster their security measures, ensuring that they are equipped to handle the unique challenges posed by these models. Furthermore, when integrating or using LLMs, preference should be given to models from established and reputable sources to ensure reliability and security. Open discussions between stakeholders, including developers, users, and regulatory bodies, can foster a more responsible and ethical adoption of Generative AI tools in software engineering. By addressing these concerns, we can ensure a balanced and responsible approach to the integration of Generative AI tools, aligning with the insights and framework provided by our study.

This study's significance lies not only in its immediate findings but also in establishing a foundation for understanding software developers' perceptions of AI during this early phase. This insight is indispensable for guiding the design and implementation of these tools to align with user needs and expectations.

Last, while this study offers an important snapshot of the current state of Generative AI adoption in software development, a comprehensive assessment of the HACAF model necessitates long-term and longitudinal studies. Such investigations would facilitate a more profound understanding of the evolution and interplay of the factors affecting adoption as the technology matures and gains wider adoption. Longitudinal studies will provide nuanced insights into changing adoption patterns, further refining the HACAF model over time, and contributing to a more sophisticated understanding of technology adoption phenomena.

7.1 Implications

The findings of this study have far-reaching implications for practitioners and researchers in software engineering and AI, offering new insights from the perspective of HACAF. The integration of AI, specifically Generative AI tools like LLMs, into software development processes has shown promise, leading to potential advancements in various aspects of the field.

A key implication of our study is the need for organizations to consider investing in AI-driven tools that fit well within existing development workflows. To translate this into actionable steps, we recommend the following:

- *Tool evaluation*: Organizations should initiate a thorough evaluation of available AI-driven tools, focusing on their compatibility with current processes and the specific needs of their development teams.
- *Pilot testing*: Before a full-scale implementation, conduct pilot tests with a sub-set of projects to gauge the tool's impact on development time, software quality, and user satisfaction.
- *Training programs*: Introduce training sessions for developers to familiarize them with the nuances of the selected AI tools, ensuring that they can leverage the tool's capabilities to the fullest.
- *Feedback loop*: Establish a feedback mechanism where developers can report on the tool's performance, any challenges faced, and areas of improvement. This feedback can guide iterative refinements and ensure the tool remains aligned with evolving development practices.
- *Cost-benefit analysis*: Regularly assess the return on investment from the AI tool adoption, considering factors like improved efficiency, reduced errors, and user satisfaction.

Given our findings on the importance of Compatibility Factors in driving AI adoption, there is an urgent need for continuous education and training in AI as it becomes more prevalent in software engineering. Concretely, we suggest the following:

- *Tailored training modules*: Organizations should develop and offer tailored training modules that focus on the integration of AI tools within existing software development workflows.

These modules should address both the technical and collaborative aspects of using AI in team settings.
— *Regular workshops*: Host regular workshops and seminars featuring experts in the field of AI. This will provide employees with insights into the latest advancements and best practices in AI-driven software development.
— *Collaboration with AI tool providers*: Forge partnerships with AI tool providers to facilitate hands-on training sessions, ensuring that the workforce is adept at leveraging the full potential of these tools.
— *Strengthening industry-academic synergy*: To bridge the gap between theoretical knowledge and practical application, academic institutions should foster deeper collaborations with industry leaders. By co-designing software engineering curricula that emphasize AI-driven development, we can ensure that graduates are not only well versed in current technologies but also attuned to the real-world challenges and opportunities of the industry.
— *Feedback-driven iterations*: Establish mechanisms to gather feedback from employees undergoing training. This feedback can be instrumental in refining training content and methodologies, ensuring that they remain relevant and effective.

Our study also underscores the significance of the human-in-the-loop approach when incorporating AI in software engineering. To use the full potential of AI tools while ensuring that they augment, rather than replace, human capabilities, the following actionable steps are recommended:

— *User-centric AI design*: AI tool developers should prioritize a user-centric design philosophy, ensuring that the tools are intuitive and seamlessly integrate into the developers' workflow.
— *Transparency mechanisms*: Implement mechanisms within AI systems that elucidate their decision-making processes. This will empower developers to understand and trust the AI's suggestions and decisions.
— *Calibration features*: Equip AI tools with features that allow developers to calibrate or fine-tune them based on specific project requirements and their professional judgment.
— *Feedback loops*: Establish iterative feedback loops where developers can provide feedback on AI tool outputs, which can then be used to refine and improve the tool's performance over time.
— *Collaborative workspaces*: Design collaborative workspaces where both AI tools and developers can contribute in real time, fostering a symbiotic relationship that leverages the strengths of both.
— *Continuous training*: As AI tools evolve, provide developers with continuous training opportunities to ensure they can effectively collaborate with the latest versions of these tools.

Importantly, the successful deployment of AI in software engineering relies heavily on aligning AI techniques with the unique needs and contexts of each organization. A careful assessment of requirements, resources, and constraints should precede the adoption of any AI-driven solution. To ensure that AI tools and techniques are not just adopted but also effectively integrated, the following steps are crucial:

— *Requirement analysis*: Before diving into AI adoption, organizations should conduct a thorough analysis of their specific needs. This involves understanding the challenges they face and identifying how AI can address them.
— *Resource evaluation*: Assess the available resources, both in terms of hardware and human expertise. This will help in selecting AI solutions that the organization has the capacity to deploy and maintain.
— *Constraint identification*: Recognize any constraints, be they budgetary, temporal, or technical, that might impact the deployment and operation of AI solutions.

— *Feedback mechanism*: Establish a mechanism for developers and other stakeholders to provide feedback on the AI tools, ensuring that they are refined and optimized over time.
— *Continuous review*: Periodically review the AI adoption strategy to ensure that it remains aligned with the evolving needs and contexts of the organization.

Our research, while primarily centered on the adoption dynamics of Generative AI tools in software engineering, did highlight the ethical implications and potential risks associated with their use. LLMs, despite their capabilities, can sometimes produce outputs that might be considered harmful or inadvertently mirror biases from their training data. Such unintended outputs can have profound implications, especially when integrated into software products for a vast audience:

— *Security vulnerabilities*: Recent studies, such as the one by Tsamados et al. [106], have shed light on the security vulnerabilities inherent to AI, particularly with LLMs. Their findings underscore the need for a thorough understanding of these models' capabilities and vulnerabilities.
- *Model provider responsibility*: Model providers, in their quest to foster adoption, should be proactive in detailing potential vulnerabilities and offering guidelines on best practices to counteract these risks. As LLM technologies evolve rapidly, periodic reviews and audits are paramount to ensure transparency and adherence to ethical standards.
- *Platform security measures*: Platforms integrating LLMs should amplify their security protocols, ensuring they are prepared to tackle the unique challenges presented by these models. When adopting or utilizing LLMs, it is prudent to choose models from well-established and trustworthy sources, guaranteeing both dependability and security.
- *Stakeholder dialogues*: Open conversations among stakeholders, encompassing developers, users, and regulatory entities can champion a more responsible and ethical adoption of Generative AI tools in software engineering.

In summary, the integration of AI techniques, especially Generative AI tools like LLMs, in software engineering holds immense potential for enhancing the development process and improving overall software quality. By acknowledging and addressing the challenges associated with AI adoptiontaking into consideration our findings on the importance of Compatibility Factorsorganizations can effectively leverage AI-driven tools and methodologies to realize enhanced outcomes in their software development pursuits.

7.2 Limitations

Our threats to validity are deliberated in relation to the application of both qualitative and quantitative validity frameworks, as advised by earlier research [86, 87]. Consequently, we commence our discourse on the credibility, transferability, and confirmability for qualitative examination [45].

Credibility. The principal variables of our investigation were informed by the three core theoretical models that assess individual, technological, and social-level influences. These models are TAM, DOI, and SCT. By integrating these thoroughly validated theories into our study, we could deeply understand the factors influencing language model adoption and further probe how these variables are implemented within the software engineering domain. Moreover, to ensure the credibility of our data, informants underwent a rigorous multi-stage selection process, verifying their roles as software engineers actively working with Generative AI tools. This meticulous selection process fortified the integrity of our study and the resulting insights.

Transferability and Confirmability. Our qualitative data analysis was performed by a single researcher, applying the Gioia methodology to the collected data. The utilization of a single researcher for data analysis could be perceived as a limitation due to potential biases and subjectivity. However, employing the Gioia methodology significantly mitigates these biases, contributing

to the trustworthiness of our findings. The Gioia methodology provides a structured approach that emphasizes transparency, iterative categorization, and systematic data processing, which lends itself to limiting subjective bias [38]. Moreover, our research merged qualitative data with a sample study to bolster the transferability of our findings. Transferability, as defined in qualitative research, refers to the extent to which the results can be applied in other contexts or with other respondents [26]. By incorporating a sample study, we provided a broader base from which parallels could be drawn, thereby enhancing the applicability of our research to varied contexts. Despite surveying a broad group of participants, limitations were present as we were unable to conduct follow-up inquiries. This may potentially affect the depth and comprehensiveness of our data. Yet, our utilization of the Gioia methodology and the combination of qualitative and sample study data have fortified the structure and interpretative robustness of our data.

Furthermore, we discuss statistical conclusion, internal, construct, and external validity to assess the quantitative investigation [117].

Internal. Our research model was validated using a cluster-randomized probability sampling strategy [43]. Because of feasibility issues, we selected a cluster of the global population (i.e., the Prolific community) instead of the entire population. Although less accurate than random sampling, cluster sampling is more cost-efficient. In response to the call of Baltes and Ralph [8], which noted that less than 10% of software engineering studies in top venues utilize probability sampling, we designed our study accordingly. Our data quality was boosted by a multi-stage process in which only 184 carefully selected professionals were chosen out of the 831 potential candidates identified (approximately 22% of the initial candidates). Nevertheless, our sample is not representative of the software engineering population.

External. The generalization of our findings was a significant concern during the PLS-SEM analysis, as sample studies are best suited for theory generalization [98]. We gathered 184 responses, an ample size considering the *a priori* power study we conducted prior to data collection.

Construct. Constructs were gauged via a single-informant approach, embodying a software engineer's viewpoint. Additionally, we employed self-reported measures, asking participants to express their agreement level on literature-derived indicators. However, these questions might not have been accurately answered. To counteract these limitations, we introduced three random attention checks, which 11 candidates failed. Furthermore, we adjusted our measurement instrument based on pre-existing ones. Last, we randomized the questionnaire and tested it for clarity and consistency to manage potential accuracy biases.

Statistical Conclusion. We processed the survey results using PLS-SEM with the renowned statistical software SmartPLS (4.0.9.5), which has been utilized in more than 1,000 peer-reviewed articles [84]. All statistical methods and tests employed for the PLS-SEM analysis are in line with the most recent guidelines in our field [91].

8 CONCLUSION

This study presented a mixed-methods investigation about the adoption of Generative AI tools within the field of software engineering. By developing HACAF, our research provided a nuanced understanding of the complexities involved in the adoption of such technologies, particularly during these nascent stages of the Generative AI transformation.

Our findings shed light on the pivotal role of Compatibility Factors, emphasizing the need for AI tools to fit within existing development workflows to enhance their adoption. This points toward the understanding that adoption is not driven solely by the perceived benefits of the technology, but by its seamless integration into the user's pre-established work processes.

Beyond its theoretical contributions, this study has significant practical implications. It offers early insights into software developers' perceptions of AI, providing valuable pointers for the

design and refinement of user-focused AI tools. These insights can help foster a more widespread adoption of AI tools by addressing developer concerns and optimizing tool compatibility with existing workflows.

As we look forward, future research in the nascent field of AI and software engineering can build upon the foundation laid by this study. While this study delivers an important snapshot of the current state of Generative AI adoption in software development, it is crucial to recognize that a comprehensive assessment of the HACAF model necessitates long-term and longitudinal studies. Such investigations would permit a deeper understanding of the evolution and interplay of factors influencing adoption as the technology matures and gains broader acceptance. These longitudinal studies would offer nuanced insights into the changing adoption patterns, thus allowing the continuous refinement of the HACAF model and contributing to a more sophisticated understanding of technology adoption phenomena.

SUPPLEMENTARY MATERIALS

The PLS-SEM computational tables, raw data, the survey instruments, and the overfitting analysis are openly available under a CC BY 4.0 license on Zenodo (DOI: https://doi.org/10.5281/zenodo.8124332).

APPENDICES

A APPENDIX A (SURVEY INSTRUMENT)

Table 20. Items Description (Those Prefixed with an Asterisk (*) Were Dropped Because of Their Insufficient Loading onto Their Latent Variable)

Construct	Item ID	Question	References
Perceptions about the Technology	PT_1	Using LLMs in my job enables me to accomplish tasks more quickly.	[29, 111]
	PT_2	Using LLMs would improve my job performance.	[29, 111]
	PT_3	Using LLMs in my job would increase my productivity.	[29, 111]
	PT_4	Using LLMs would enhance my effectiveness on the job.	[29, 111]
	PT_5	(*) Using LLMs would make it easier to do my job.	[29, 111]
	PT_6	I would find LLMs useful in my job.	[29, 111]
	PT_7	I would find LLMs easy to use.	[29, 111]
	PT_8	(*) It would be easy for me to become skillful at using LLMs.	[29, 111]
	PT_9	(*) I would find LLMs flexible to interact with.	[29, 111]
	PT_10	(*) Learning to operate LLMs would be easy for me.	[65, 111]
	PT_11	Using LLMs would enable me to accomplish tasks more quickly.	[29, 111]
	PT_12	Using LLMs would improve the quality of work I do.	[65, 111]
	PT_13	Using LLMs would make it easier to do my job.	[65, 111]
	PT_14	Using LLMs would enhance my effectiveness on the job.	[65, 111]
	PT_15	(*) Using a LLM gives me greater control over my work.	[4, 111]
Compatibility Factors	CF_1	(*) Using LLMs is compatible with all aspects of my work.	[65, 111]
	CF_2	I think that using a LLM fits well with the way I like to work.	[65, 111]
	CF_3	Using a LLM fits into my work style.	[65, 111]
	CF_4	(*) I have control over using LLMs.	[4, 111]
	CF_5	I have the knowledge necessary to use LLMs.	[4, 111]

(Continued)

Table 20. Continued

	CF_6	Given the resources, opportunities, and knowledge it takes to use the LLMs, it would be easy for me to use LLMs.	[4, 111]
Social Factors	SF_1	People who influence my behaviour think that I should use LLMs.	[4, 111]
	SF_2	People who are important to me think that I should use LLMs.	[4, 111]
	SF_3	(*) I use LLMs because of the proportion of coworkers who use the system.	[102, 111]
	SF_4	(*) People in my organisation who use LLMs have more prestige than those who do not.	[65, 111]
Personal and Environmental Factors	PEF_1	(*) If I heard about a new technology like LLMs, I would look for ways to experiment with it.	[111]
	PEF_2	(*) Among my peers, I am usually the first to try out new technologies like LLMs.	[1]
	PEF_3	(*) I like to experiment with new technologies.	[1]
	PEF_4	My organization provides the necessary resources for using LLMs effectively.	[55, 111]
	PEF_5	My organization offers sufficient training sessions to enhance our skills in using LLMs.	[55, 111]
	PEF_6	My organization encourages the use of LLMs in our daily work.	[56, 111]
	PEF_7	I feel that there is top management support in my organization for using LLMs.	[56, 111]
Intention to Use	IU_1	I intend to use LLMs more extensively in the next months.	[29, 111]
	IU_2	I predict I would use LLMs more extensively in the next months.	[29, 111]
	IU_3	I plan to use LLM more extensively in the next months.	[29, 111]

B APPENDIX B (OVERFITTING ANALYSIS)

The notably high effect size observed between 'Perception about the Technology' and 'Compatibility Factors' might raise concerns of potential overfitting, a scenario where the model inadvertently learns noise in the training data, thereby compromising its ability to accurately predict unseen data [47]. Given the potential implications of overfitting on the reliability of our model, it is crucial to thoroughly investigate this issue. The details of this analysis, along with the associated code and data, can be found in the online supplementary materials hosted on Zenodo.[4]

First, we analyzed the residuals of our model, which can provide insights into the appropriateness of the model fit. The residuals represent the difference between the observed and predicted values for the dependent variable. In a well-specified model, we would expect the residuals to be randomly scattered around zero, with no apparent pattern. This would suggest that the model's errors are random, and that the model is correctly specified.

We plotted the residuals against the predicted values for the 'Intention to Use' construct and found that they were indeed mostly randomly scattered, suggesting that our model's errors are random. Figure 4 shows this residuals plot.

However, to evaluate the risk of overfitting more thoroughly, we employed two key methodologies: a train-test split and cross validation [53]. In the train-test split, we partitioned our data into a training set (80% of the data) and a testing set (20% of the data). Our model was trained on the training data and then evaluated on the unseen testing data.

[4]Link to the replication package: https://doi.org/10.5281/zenodo.8124332

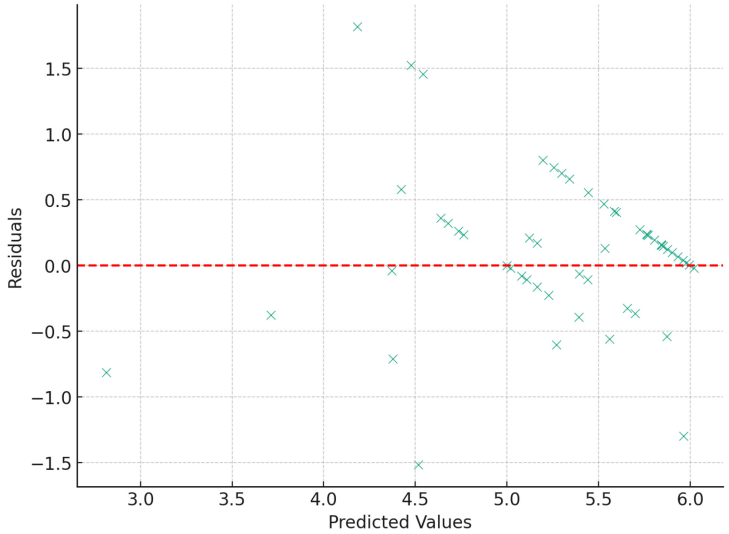

Fig. 4. Residuals vs. predicted values for the 'Intention to Use' construct. The residuals are mostly randomly scattered around zero, suggesting that the model's errors are random and that the model is correctly specified.

The model's performance was assessed using the **mean squared error (MSE)**, a metric that calculates the average squared difference between the observed and predicted values. A lower MSE indicates a better fit to the data. The MSE for the test set was approximately 0.523.

To further examine overfitting, we implemented k-fold cross validation with k set to 5, a common choice for this parameter [53]. In this approach, the data was divided into five sub-sets, and the model was trained and tested five times, each time on a different sub-set of the data. The performance of the model was again assessed using MSE. The mean MSE from the cross validation was approximately 0.676, reasonably close to the test MSE, suggesting that our model is not overfitting the data.

In conclusion, both the train-test split and cross-validation results suggest that our model is generalizing effectively to unseen data and is not overfitting.

ACKNOWLEDGMENT

ChatGPT-4 has been used to ensure linguistic accuracy and enhance the readability of this article.

REFERENCES

[1] Ritu Agarwal and Jayesh Prasad. 1998. A conceptual and operational definition of personal innovativeness in the domain of information technology. *Information Systems Research* 9, 2 (1998), 204–215.

[2] Ritu Agarwal and Jayesh Prasad. 1999. Are individual differences germane to the acceptance of new information technologies? *Decision Sciences* 30, 2 (1999), 361–391.

[3] BLACKBOX AI. 2023. BLACKBOX AI—useblackbox.io. Retrieved October 17, 2023 from https://www.useblackbox.io/

[4] Icek Ajzen. 1991. The theory of planned behavior. *Organizational Behavior and Human Decision Processes* 50, 2 (1991), 179–211.

[5] Icek Ajzen. 2020. The theory of planned behavior: Frequently asked questions. *Human Behavior and Emerging Technologies* 2, 4 (2020), 314–324.

[6] Amazon Web Services. 2023. CodeWhisperer. Retrieved October 17, 2023 from https://aws.amazon.com/es/codewhisperer/

[7] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. 2020. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion* 58 (2020), 82–115.
[8] S. Baltes and P. Ralph. 2020. Sampling in software engineering research: A critical review and guidelines. *arXiv:2002.07764* (2020).
[9] Albert Bandura. 2014. Social cognitive theory of moral thought and action. In *Handbook of Moral Behavior and Development*. Psychology Press, 69–128.
[10] Albert Bandura, W. H. Freeman, and Richard Lightsey. 1999. Self-efficacy: The exercise of control. *Journal of Cognitive Psychotherapy* 13, 2 (1999), 158–166.
[11] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7 (2023), 85–111.
[12] James Bessen. 2019. Automation and jobs: When technology boosts employment. *Economic Policy* 34, 100 (2019), 589–626.
[13] Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. 2022. Taking flight with Copilot: Early insights and opportunities of AI-powered pair-programming tools. *Queue* 20, 6 (2022), 35–57.
[14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Chrisopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems* 33 (2020), 1877–1901.
[15] Javier Cámara, Javier Troya, Lola Burgueño, and Antonio Vallecillo. 2023. On the assessment of generative AI in modeling tasks: An experience report with ChatGPT and UML. *Software and Systems Modeling* 22 (2023), 781–793.
[16] Ruijia Cheng, Ruotong Wang, Thomas Zimmermann, and Denae Ford. 2022. "It would work for me too": How online communities shape software developers' trust in AI-powered code generation tools. *arXiv preprint arXiv:2212.03491* (2022).
[17] W. W. Chin. 1998. The partial least squares approach to structural equation modeling. In *Modern Methods for Business Research*. Lawrence Erlbaum Associates, 295–336.
[18] Clayton M. Christensen. 1997. *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*. Harvard Business Review Press.
[19] Michael Chui, Eric Hazan, Roger Roberts, Alex Singla, Kate Smaje, Alex Sukharevsky, Lareina Yee, and Rodney Zemmel. 2023. The economic potential of generative AI: The next productivity frontier. Retrieved July 3, 2023 from https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier
[20] Victoria Clarke, Virginia Braun, and Nikki Hayfield. 2015. Thematic analysis. In *Qualitative Psychology: A Practical Guide to Research Methods*, Jonathan A. Smith (Ed.). SAGE Publications, 222–248.
[21] CodeComplete. 2023. AI Coding Assistant for Enterprise. Retrieved October 17, 2023 from https://codecomplete.ai/
[22] Codeium. 2023. Free AI Code Completion and Chat. Retrieved October 17, 2023 from https://codeium.com/
[23] J. Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences*. Routledge.
[24] Deborah R. Compeau and Christopher A. Higgins. 1995. Computer self-efficacy: Development of a measure and initial test. *MIS Quarterly* 19, 2 (1995), 189–211.
[25] J. Corbin and A. Strauss. 1990. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology* 13, 1 (1990), 3–21.
[26] J. Creswell. 2013. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE.
[27] Arghavan Moradi Dakhel, Vahid Majdinasab, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Zhen Ming Jack Jiang. 2023. GitHub Copilot AI pair programmer: Asset or liability? *Journal of Systems and Software* 203 (2023), 111734.
[28] Anastasia Danilova, Alena Naiakshina, Stefan Horstmann, and Matthew Smith. 2021. Do you really code? Designing and evaluating screening questions for online surveys with programmers. In *Proceedings of the International Conference on Software Engineering*. IEEE, 537–548.
[29] F. Davis. 1989. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly* 13, 3 (1989), 319–340.
[30] Fred D. Davis, Richard P. Bagozzi, and Paul R. Warshaw. 1989. User acceptance of computer technology: A comparison of two theoretical models. *Management Science* 35, 8 (1989), 982–1003.

[31] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2019).
[32] Paul Dourish. 2003. The appropriation of interactive technologies: Some lessons from placeless documents. *Computer Supported Cooperative Work* 12 (2003), 465–490.
[33] Christof Ebert and Panos Louridas. 2023. Generative AI for software practitioners. *IEEE Software* 40, 4 (2023), 30–38.
[34] Neil A. Ernst and Gabriele Bavota. 2022. AI-driven development is here: Should you worry? *IEEE Software* 39, 2 (2022), 106–110.
[35] Carl Benedikt Frey and Michael A. Osborne. 2017. The future of employment: How susceptible are jobs to computerisation? *Technological Forecasting and Social Change* 114 (2017), 254–280.
[36] Yanjie Gao, Xiaoxiang Shi, Haoxiang Lin, Hongyu Zhang, Hao Wu, Rui Li, and Mao Yang. 2023. An empirical study on quality issues of deep learning platform. In *Proceedings of the International Conference on Software Engineering: Software Engineering in Practice*. IEEE, 455–466.
[37] D. Gefen, D. Straub, and M.-C. Boudreau. 2000. Structural equation modeling and regression: Guidelines for research practice. *Communications of the Association for Information Systems* 4, 1 (2000), 7.
[38] D. Gioia, K. Corley, and A. Hamilton. 2013. Seeking qualitative rigor in inductive research: Notes on the Gioia methodology. *Organizational Research Methods* 16, 1 (2013), 15–31.
[39] D. Gioia, J. Thomas, S. Clark, and K. Chittipeddi. 1994. Symbolism and strategic change in academia: The dynamics of sensemaking and influence. *Organization Science* 5, 3 (1994), 363–383.
[40] B. Glaser and A. Strauss. 2017. *Discovery of Grounded Theory: Strategies for Qualitative Research*. Routledge.
[41] Dale L. Goodhue and Ronald L. Thompson. 1995. Task-technology fit and individual performance. *MIS Quarterly* 19, 2 (1995), 213–236.
[42] Roberto Gozalo-Brizuela and Eduardo C. Garrido-Merchán. 2023. A survey of generative AI applications. *arXiv preprint arXiv:2306.02781* (2023).
[43] F. J. Gravetter and L.-A. B. Forzano. 2018. *Research Methods for the Behavioral Sciences*. Cengage Learning.
[44] Stine Grodal, Michel Anteby, and Audrey L. Holm. 2021. Achieving rigor in qualitative analysis: The role of active categorization in theory building. *Academy of Management Review* 46, 3 (2021), 591–612.
[45] E. Guba. 1981. Criteria for assessing the trustworthiness of naturalistic inquiries. *Educational Communication and Technology Journal* 29, 2 (1981), 75–91.
[46] J. F. Hair Jr., G T. M. Hult, C. Ringle, and M. Sarstedt. 2016. *A Primer on Partial Least Squares Structural Equation Modeling (PLS-SEM)*. SAGE.
[47] Douglas M. Hawkins. 2004. The problem of overfitting. *Journal of Chemical Information and Computer Sciences* 44, 1 (2004), 1–12.
[48] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751* (2019).
[49] Saki Imai. 2022. Is GitHub Copilot a substitute for human pair-programming? An empirical study. In *Proceedings of the ACM/IEEE International Conference on Software Engineering: Companion Proceedings*. 319–321.
[50] L. Isabella. 1990. Evolving interpretations as a change unfolds: How managers construe key organizational events. *Academy of Management Journal* 33, 1 (1990), 7–41.
[51] Mateusz Jaworski and Dariusz Piotrkowski. 2023. Study of software developers' experience using the GitHub Copilot Tool in the software development process. *arXiv:2301-04991* (2023).
[52] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).
[53] Ron Kohavi. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, Vol. 14. 1137–1145.
[54] Rajiv Kohli and Nigel P. Melville. 2019. Digital innovation: A review and synthesis. *Information Systems Journal* 29, 1 (2019), 200–223.
[55] U. Kulkarni, S. Ravindran, and R. Freeze. 2006. A knowledge management success model: Theoretical development and empirical validation. *Journal of Management Information Systems* 23, 3 (2006), 309–347.
[56] W. Lewis, R. Agarwal, and V. Sambamurthy. 2003. Sources of influence on beliefs about information technology use: An empirical study of knowledge workers. *MIS Quarterly* 27, 4 (2003), 657–678.
[57] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, T. Hubert, P. Choy, C. de Masson d'Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, and O. Vinyals. 2023. AlphaCode Attention Visualization. Retrieved October 17, 2023 from https://alphacode.deepmind.com/
[58] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2023. Understanding the usability of AI programming assistants. *arXiv preprint arXiv:2303.17125* (2023).

[59] Y. Lincoln and E. Guba. 1985. *Naturalistic Inquiry*. SAGE.
[60] Mai Skjott Linneberg and Steffen Korsgaard. 2019. Coding qualitative data: A synthesis guiding the novice. *Qualitative Research Journal* 19, 3 (2019), 259–270.
[61] K. Locke. 1996. Rewriting the discovery of grounded theory after 25 years? *Journal of Management Inquiry* 5, 3 (1996), 239–245.
[62] Antonio Mastropaolo, Luca Pascarella, Emanuela Guglielmi, Matteo Ciniselli, Simone Scalabrino, Rocco Oliveto, and Gabriele Bavota. 2023. On the robustness of code generation techniques: An empirical study on GitHub Copilot. *arXiv preprint arXiv:2302.00438* (2023).
[63] J. Miles. 2014. *Tolerance and Variance Inflation Factor*. American Cancer Society.
[64] Brent Daniel Mittelstadt, Patrick Allo, Mariarosaria Taddeo, Sandra Wachter, and Luciano Floridi. 2016. The ethics of algorithms: Mapping the debate. *Big Data & Society* 3, 2 (2016), 1–21.
[65] Geoffrey A. Moore and Regis McKenna. 1999. *Crossing the Chasm*. HarperBusiness.
[66] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2022. Reading between the lines: Modeling user behavior and costs in AI-assisted programming. *arXiv preprint arXiv:2210.14306* (2022).
[67] Mutable.ai. 2023. AI Accelerated Software Development. Retrieved October 17, 2023 from https://mutable.ai/
[68] Nhan Nguyen and Sarah Nadi. 2022. An empirical evaluation of GitHub Copilot's code suggestions. In *Proceedings of the International Conference on Mining Software Repositories*. 1–5.
[69] Oded Nov and Chen Ye. 2008. Users' personality and perceived ease of use of digital libraries: The case for resistance to change. *Journal of the American Society for Information Science and Technology* 59, 5 (2008), 845–851.
[70] J. Nunnally. 1978. *Psychometric Methods*. McGraw-Hill.
[71] World Medical Association. 2014. World Medical Association Declaration of Helsinki: Ethical principles for medical research involving human subjects. *Journal of the American College of Dentists* 81, 3 (2014), 14.
[72] OpenAI. 2023. GPT-4 technical report. *arXiv:2303.08774 [cs.CL]* (2023).
[73] Ipek Ozkaya. 2023. The next frontier in software development: AI-augmented software development processes. *IEEE Software* 40, 4 (2023), 4–9.
[74] Stefan Palan and Christian Schitter. 2018. Prolific.acA subject pool for online experiments. *Journal of Behavioral and Experimental Finance* 17 (2018), 22–27.
[75] Sida Peng, Eirini Kalliamvakou, Peter Cihon, and Mert Demirer. 2023. The impact of AI on developer productivity: Evidence from GitHub Copilot. *arXiv preprint arXiv:2302.06590* (2023).
[76] Anthony Peruma, Steven Simmons, Eman Abdullah AlOmar, Christian D. Newman, Mohamed Wiem Mkaouer, and Ali Ouni. 2022. How do I refactor this? An empirical study on refactoring trends and topics in Stack Overflow. *Empirical Software Engineering* 27, 1 (2022), 11.
[77] James Prather, Brent N. Reeves, Paul Denny, Brett A. Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. "It's weird that it knows what I want": Usability and interactions with CoPilot for novice programmers. *arXiv preprint arXiv:2304.02491* (2023).
[78] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog* 1, 8 (2019), 9.
[79] Paul Ralph, Nauman bin Ali, Sebastian Baltes, Domenico Bianculli, Jessica Diaz, Yvonne Dittrich, Neil Ernst, Michael Felderer, Robert Feldt, Antonio Filieri, Breno Bernard Nicolau de Franca, Carlo Alberto Furia, Greg Gay, Nicolas Gold, Daniel Graziotin, Pinjia He, Rashina Hoda, Natalia Juristo, Barbara Kitchenham, Valentina Lenarduzzi, Jorge Martinez, Jorge Melegati, Daniel Mendez, Tim Menzies, Jefferson Molleri, Dietmar Pfahl, Romain Robbes, Daniel Russo, Nyyti Saarimaki, Federica Sarro, Davide Taibi, Janet Siegmund, Diomidis Spinellis, Miroslaw Staron, Klaas Stol, Margaret-Anne Storey, Damian Tamburri, Marco Torchiano, Christoph Treude, Burak Turhan, Xiaofeng Wang, and Sira Vegas. 2020. Empirical standards for software engineering research. *arXiv preprint arXiv:2010.03525* (2020).
[80] M. Ramkumar, T. Schoenherr, S. Wagner, and M. Jenamani. 2019. Q-TAM: A quality technology acceptance model for predicting organizational buyers' continuance intentions for e-procurement services. *International Journal of Production Economics* 216 (2019), 333–348.
[81] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A large scale study of programming languages and code quality in GitHub. In *Proceedings of the International Symposium on Foundations of Software Engineering*. ACM, 155–165.
[82] Replit. 2023. Ghostwriter—Code Faster with AI. Retrieved October 17, 2023 from https://replit.com/site/ghostwriter
[83] C. M. Ringle and M. Sarstedt. 2016. Gain more insight from your PLS-SEM results: The importance-performance map analysis. *Industrial Management & Data Systems* 116, 9 (2016), 1865–1886.
[84] C. M. Ringle, S. Wende, and J.-M. Becker. 2015. *SmartPLS 3*. SmartPLS GmbH, Boenningstedt.
[85] Everett M. Rogers. 2010. *Diffusion of Innovations*. Simon & Schuster.
[86] Daniel Russo. 2021. The agile success model: A mixed-methods study of a large-scale agile transformation. *ACM Transactions on Software Engineering and Methodology* 30, 4 (2021), 1–46.

[87] D. Russo, P. Ciancarini, T. Falasconi, and M. Tomasi. 2018. A meta model for information systems quality: A mixed-study of the financial sector. *ACM Transactions on Management Information Systems* 9, 3 (2018), 1–38.
[88] Daniel Russo, Paul H. P. Hanel, Seraphina Altnickel, and Niels van Berkel. 2021. The daily life of software engineers during the covid-19 pandemic. In *Proceedings of the International Conference on Software Engineering*. IEEE, 364–373.
[89] Daniel Russo, Paul H. P. Hanel, Seraphina Altnickel, and Niels van Berkel. 2021. Predictors of well-being and productivity among software professionals during the COVID-19 pandemic—A longitudinal study. *Empirical Software Engineering* 26, 62 (2021), 1–64. https://doi.org/10.1007/s10664-021-09945-9
[90] Daniel Russo and Klaas-Jan Stol. 2020. Gender differences in personality traits of software engineers. *IEEE Transactions on Software Engineering* 48, 3 (2020), 16.
[91] Daniel Russo and Klaas-Jan Stol. 2021. PLS-SEM for software engineering research: An introduction and survey. *ACM Computing Surveys* 54, 4 (2021), 1–38.
[92] Mahadev Satyanarayanan. 2001. Pervasive computing: Vision and challenges. *IEEE Personal Communications* 8, 4 (2001), 10–17.
[93] Albrecht Schmidt. 2023. Speeding Up the engineering of interactive systems with Generative AI. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. 7–8.
[94] A. H. Segars and V. Grover. 1993. Re-examining perceived ease of use and usefulness: A confirmatory factor analysis. *MIS Quarterly* 17, 4 (1993), 517–525.
[95] Galit Shmueli, Soumya Ray, Juan Manuel Velasquez Estrada, and Suneel Babu Chatla. 2016. The elephant in the room: Predictive performance of PLS models. *Journal of Business Research* 69, 10 (2016), 4552–4564.
[96] G. Shmueli, M. Sarstedt, J. F. Hair, J. Cheah, H. Ting, S. Vaithilingam, and C. M. Ringle. 2019. Predictive model assessment in PLS-SEM: Guidelines for using PLSpredict. *European Journal of Marketing* 53, 11 (2019), 2322–2347.
[97] Dominik Sobania, Martin Briesch, and Franz Rothlauf. 2022. Choose your programming copilot: A comparison of the program synthesis performance of GitHub Copilot and genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1019–1027.
[98] Klaas-Jan Stol and Brian Fitzgerald. 2018. The ABC of software engineering research. *ACM Transactions on Software Engineering and Methodology* 27, 3 (2018), 11.
[99] A. Strauss and J. Corbin. 1990. *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. SAGE.
[100] Tabnine. 2023. AI Assistant for Software Developers. Retrieved October 17, 2023 from https://www.tabnine.com/
[101] Emmanuel Tenakwah. 2021. What do employees want?: Halting record-setting turnovers globally. *Strategic HR Review* 20, 6 (2021), 206–210.
[102] Ronald L. Thompson, Christopher A. Higgins, and Jane M. Howell. 1991. Personal computing: Toward a conceptual model of utilization. *MIS Quarterly* 15, 1 (1991), 125–143.
[103] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F. Bissyandé. 2023. Is ChatGPT the ultimate programming assistant—How far is it? *arXiv preprint arXiv:2304.11938* (2023).
[104] Gholamreza Torkzadeh, Jerry Cha-Jan Chang, and Didem Demirhan. 2006. A contingency model of computer and Internet self-efficacy. *Information & Management* 43, 4 (2006), 541–550.
[105] Louis G. Tornatzky and Katherine J. Klein. 1982. Innovation characteristics and innovation adoption-implementation: A meta-analysis of findings. *IEEE Transactions on Engineering Management* 29, 1 (1982), 28–45.
[106] Andreas Tsamados, Luciano Floridi, and Mariarosaria Taddeo. 2023. The cybersecurity crisis of artificial intelligence: Unrestrained adoption and natural language-based attacks. *arXiv:2311.09224* (2023).
[107] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Proceedings of the International Conference on Human Factors in Computing Systems*. 1–7.
[108] J. Van Maanen. 1979. The fact of fiction in organizational ethnography. *Administrative Science Quarterly* 24, 4 (1979), 539–550.
[109] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* 30 (2017), 1–11.
[110] Viswanath Venkatesh and Fred D. Davis. 2000. A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management Science* 46, 2 (2000), 186–204.
[111] Viswanath Venkatesh, Michael G. Morris, Gordon B. Davis, and Fred D. Davis. 2003. User acceptance of information technology: Toward a unified view. *MIS Quarterly* 27, 3 (2003), 425–478.
[112] Viswanath Venkatesh, James Y. L. Thong, and Xin Xu. 2016. Unified theory of acceptance and use of technology: A synthesis and the road ahead. *Journal of the Association for Information Systems* 17, 5 (2016), 328–376.
[113] The Economist. 2023. The Widespread Adoption of AI by Companies Will Take a While. Retrieved April 5, 2024 from https://www.economist.com/leaders/2023/06/29/the-widespread-adoption-of-ai-by-companies-will-take-a-while
[114] Ruotong Wang, Ruijia Cheng, Denae Ford, and Thomas Zimmermann. 2023. Investigating and designing for trust in AI-powered code generation tools. *arXiv preprint arXiv:2305.11248* (2023).

[115] Bolin Wei, Ge Li, Xin Xia, Zhiyi Fu, and Zhi Jin. 2019. Code generation as a dual task of code summarization. In *Proceedings of the International Conference on Neural Information Processing Systems*. 6563–6573.

[116] Michel Wermelinger. 2023. Using GitHub Copilot to solve simple programming problems. In *Proceedings of the ACM Technical Symposium on Computer Science Education*. 172–178.

[117] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in Software Engineering*. Springer Science & Business Media.

[118] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. 2022. Assessing the quality of GitHub Copilot's code generation. In *Proceedings of the International Conference on Predictive Models and Data Analytics in Software Engineering*. 62–71.

[119] Beiqi Zhang, Peng Liang, Xiyu (Thomas) Zhou, Aakash Ahmad, and Muhammad Waseem. 2023. Practices and challenges of using GitHub Copilot: An empirical study. *arXiv preprint arXiv:2303.08733* (2023).

[120] Daniel Zhang, Saurabh Mishra, Erik Brynjolfsson, John Etchemendy, Deep Ganguli, Barbara Grosz, Terah Lyons, James Manyika, Juan Carlos Niebles, Michael Sellitto, Yoav Shoham, Jack Clark, and Raymond Perrault. 2021. The AI index 2021 annual report. *arXiv preprint arXiv:2103.06312* (2021).

[121] Q. Zheng, X. Xia, X. Zou, Y. Dong, S. Wang, Y. Xue, Z. Wang, L. Shen, A. Wang, Y. Li, T. Su, Z. Yang, and J. Tang. 2023. GitHub—THUDM/CodeGeeX: An Open Multilingual Code Generation Model. Retrieved October 17, 2023 from https://github.com/THUDM/CodeGeeX

Received 12 July 2023; revised 3 January 2024; accepted 18 January 2024