Coding with a Co-Pilot: The impact of LLMs on Software Developers

AMIR KUANOV, University of Twente, The Netherlands

Currently, large language models (LLMs) like GitHub Copilot and ChatGPT are widely used within software development projects to increase the productivity in daily workflows. AI-supported assistants have been shown to be quite productive tools in the hands of software engineers. This paper aims to investigate how professional software developers perceive the impact of LLMs on their problem-solving skills and independence. This qualitative study identifies patterns in cognitive offloading, changes in development approaches, and strategic adaptation by conducting semi-structured interviews with professional software developers. The findings contribute to a better understanding of the implications of LLM use on software engineering problem-solving skills, filling a gap in current productivity-focused literature.

Additional Key Words and Phrases: Large Language Models (LLMs), AI-assisted programming, Software engineering, GitHub Copilot,AI tools in software development, Human-AI collaboration, Skill retention, Problem Solving, Cognitive Offloading

1 INTRODUCTION

The integration of large language models (LLMs) into software engineering has pushed human-AI collaboration to a new level, allowing software engineers to generate and adapt code snippets through human language prompts, enable code auto completion and suggestion by connecting them into an integrated development environment (IDE), summarize and write code documentation [14, 20, 22]. A large language model is a type of deep neural network that has been trained on various text datasets, to identify patterns and relationships within a language [20]. This method allows the model to generate a grammatically correct text and structured code [16]. LLMs have been shown to be effective at code generation, bug detection and automated documentation [8, 16].

Software developer productivity has been widely studied, with factors such as perceived productivity, lines of code written, coding time, code submission and task completion time commonly used as indicators [9]. Large Language Models (LLMs), such as GitHub Copilot, are increasingly adopted in software development [13, 21], as they have the potential to influence these dimensions of productivity. By assisting with repetitive coding tasks, offering syntax and logic suggestions, LLMs help developers maintain focus and efficiency. Recent studies highlight a measurable increase in perceived productivity when LLMs are used as coding assistants [22], particularly when integrated into development environments to provide in-line support, such as code completions and automated comments.

Existing literature raises concern about the effects of long-term LLM usage on developers' problem-solving skills [4]. Problem solving is typically defined as the process of identifying a challenge, analyzing it, and determining an effective solution [15]. In software development, this often involves synthesizing knowledge, applying logic, and navigating complex technical tasks [11]. Theoretical concerns stem from the idea that if developers rely heavily on LLM-powered copilots to generate solutions, they may engage less in the critical thinking and exploratory behaviors essential for deep problem-solving.

It is established that LLM assistants are used as a tool to increase efficiency of software developers. Cognitive offloading is a psychological concept that refers to the use of external tools or agents to reduce the mental effort required for cognitive tasks. This theory is relevant in contexts where technology is used to augment or replace human thought processes [18]. In case the of software development, LLMs act as external tools to aid in cognitive processes, helping developers to recall syntax, break down and summarize code and generate code snippets [14, 20, 22]. Cognitive offloading theory suggests that tecnnological tools can enhance performance, excessive reliance may reduce the retention and application of skills over time [18]. That is why cognitive offloading serves as a suitable theory to examine how thinking patterns of software developers change while they are using LLMs in their workflow. It allows this study to understand possible negative effects on problem-solving and long-term skill retention that may come with increasing reliance on these tools.

This study proposes a qualitative approach that explores how software developers describe impact of LLM-powered copilots. By using data acquired from real world professionals, this study aims to research the impact of Large Language Models (LLMs) from software developers point of view.

2 BACKGROUND

2.1 Large Language Models in Software Engineering

The applications that are based on the LLMs, such as ChatGPT and Github Copilot, are quite often used in software development [13, 21]. Recent surveys mention that 84 percent out of 57 professional developers use these applications at least several times a week [4]. Large language models use vast datasets of source code and natural language to generate code snippets in real time [8]. While their integration into tools like GitHub Copilot has demonstrated noteworthy improvements in developer productivity , challenges such as hallucinated outputs, usability and correctness still need to be investigated [8, 14, 21].

2.2 Problem Solving Theory

Problem-solving is a cognitive activity that consists of identifying a challenge, understanding it, and taking steps to overcome it [7]. It is employed in a wide range of real-life scenarios that range from planning daily activities or more complex and significant tasks. Problem-solving is the ability to understand the gap between given state and a desired goal, and applying reasoning and learned knowledge to come up with a solution. It often requires

TScIT 43, July 4, 2025, Enschede, The Netherlands

analyzing information, researching, generating and comparing alternatives, adapting strategies and validating solutions [7]. This process is essential in both academic and professional domains, where adaptive decision-making and independent judgment are often required. It is widely regarded as one of the most transferable and teachable cognitive skills, serving as a foundation for critical thinking, and decision-making.

2.2.1 Problem Solving in Software Engineering. Problem solving in software engineering is present in multiple activities in the daily workflow - testing, debugging, creating and refactoring modules [6]. It is crucial for the developers to understand the problem, knowing where to find the solution and, if it does not exist, create one. Software engineering consists of three traditional activities: definition, design and building the product [19]. However, how does problem solving apply to these activities? For example, in definition, software engineers must help the clients to understand what they want to accomplish and what product would be best suited for their goal [19], i.e. find a suitable solution to their problem.

2.3 Cognitive Offloading Theory

Cognitive offloading refers to the act of using the body or external environment to reduce internal mental demands [18]. Common examples include writing down notes, setting digital reminders, or using GPS for navigation. Researchers have emphasized that offloading is not just a practical shortcut but a fundamental part of how humans extend cognition into the world, specifically when performing tasks related to memory, planning, or reasoning [18]. These methods help humans to work around the limits of attention and working memory, allowing them to maintain performance even under increased cognitive load.

Offloading is shaped not only by how difficult a task is, but also by how individuals evaluate their own mental abilities. People often choose to offload when they believe their memory or attention may not be reliable. This reliance is guided by metacognitive judgments, such as lack of confidence, which can lead to offloading even when it is unnecessary [18]. While cognitive offloading can improve efficiency and reduce mistakes, studies also raise concerns about potential long-term effects. Repeated reliance on external tools may reduce engagement with internal cognitive processes, potentially weakening memory recall or independent reasoning over time. These risks are especially relevant in modern environments where intelligent tools such as AI assistants are frequently used. In fields like software engineering, where reasoning, planning, and problem-solving play a critical role, increasing reliance on intelligent tools may not only reshape development practices but also reduce the cognitive engagement traditionally required of developers.

2.4 Current State of Research

Research has shown that these tools offer productivity benefits for developers, especially in automating repetitive tasks, accelerating idea generation, and supporting routine coding work [2, 14, 22]. Recent work has conducted a grounded theory study with professionals in software development and identified clear advantages in development time, as well as during the prototyping, code analysis and planning stages. LLMs were frequently used for ideation, documentation, prototyping and error identification allowing developers to reallocate cognitive effort to higher-level tasks [2].

At the same time, related literature raised concerns about how extensive AI assistance may affect critical thinking and cognitive independence. Recent mixed-method study of 666 participants found a negative correlation between frequent AI tool use and critical thinking performance, caused by increased cognitive offloading. The findings suggest that while AI can reduce cognitive load and improve task efficiency, it may simultaneously decline problem-solving skills and independent reasoning [10]. These findings intersect with cognitive offloading theory, which states that individuals externalize mental tasks to tools in order to manage cognitive load. Although offloading can be functional in the short term, it may lead to skill decay when users become dependent on tools that bypass deeper engagement [18]. In software engineering contexts, this tension becomes especially relevant. As developers rely more on AI to generate code, review designs, or troubleshoot issues, they risk reducing their involvement in complex reasoning tasks.

2.5 Knowledge Gap

While existing work demonstrates that LLM-based tools enhance developer productivity [2, 22] and that heavy reliance on AI can impair critical thinking via cognitive offloading [10], there remains a lack of research that theoretically integrates cognitive offloading within professional software-engineering workflows, and examines how prolonged LLM use affects developers' long-term problem-solving skills and autonomy. This study addresses these gaps by applying Cognitive Offloading Theory to developers' experiences with LLM-based assistants, investigating both perceived benefits and potential cognitive costs over time.

2.6 Problem Statement

The use of LLM–based copilots improved efficiency of software engineers by automating routine tasks such as writing boilerplate code, suggesting API usages, and streamlining debugging workflows [14]. While these tools evidently boost short-term productivity and help developers prototype features in a shorter amount of time, there is growing concern that reliance on machine-generated suggestions may affect problem-solving, independence and technical learning of the software developers [4]. To address this concern and the gap in the productivity-focused literature, this study aims to answer the following research question: **How do software engineers perceive the impact of large language models on their problem solving skills and independence?**

3 METHODOLOGY

This study employs a qualitative research approach using semi-structured interviews to explore the effect of LLM-based assistants on professional software developers. A qualitative method will allow for effective capture of detailed narratives that reflect developers real experiences and perceptions, which as opposed to quantitative performance metrics [1]. Because, to answer the research question, it is important to understand how developers perceive the effects on their problem-solving skills. By sharing their experience with

the LLMs, developers show how they approach and tackle certain problems, share details of their own findings on the matter and give the understanding of the way they interact with LLMs and how they think it affects them. That is why the qualitative method allows us to understand the perception of the software developer on the matter, since quantitative metrics would not provide thought processes during the LLMs usage, real examples from their experience and their opinions on the topic.

3.1 Data Collection

Participants were selected through purposive sampling since it fits to the qualitative research approach employed in this paper [1, 12]. The sample consists of 5 professional software developers who actively use LLM-based assistants in their daily work routine. The sample is relatively small due to the time limitations for this research, however it will allow to collect reasonable insights from expert developers.

Table 1. Overview of the interviewees

ID	Industry	Role	Experience
1	Software	Full Stack Developer	2 years
2	Software	Senior Developer	13 years
3	Software	Backend Developer	11 years
4	Finance	Application Administrator	12 years
5	E-Commerce	Backend Developer	3 years

Each interview had the duration of 20 minutes on average and was conducted online. The duration was enough to collect meaningful data and avoid fatigue [5, 17]. The interviews were recorded with consent and transcribed for analysis. Questions were open-ended to encourage participants to share their views freely while following a general interview guide focused on problem-solving [1]. Interview transcriptions and recordings were stored and used securely and the data from the interviews was properly destroyed after their intended use to comply with the General Data Protection Regulation, since the research was performed in the Netherlands, which is a part of the European Union where GDPR is legally binding.

3.2 Data analysis

For the purpose of the research and extracting meaningful data from the transcribed interviews thematic analysis was performed. Thematic analysis is widely used for qualitative research and considered to be one of the ways to create meaningful insights from raw data [3]. Thematic analysis assists in identifying patterns and provides detailed description of the data that was derived from the semi-structured interviews [3]. This method helps to interpret how developers utilize AI tools, how they perceive changes in their skills, and how they differentiate the AI with independent thinking, since it organizes and describes ambiguous data without losing detail [3]. The analysis consists of 6 phases:

4 RESULTS

This section is presented as a continuous narrative. After coding and analyzing data across five interviews with professional software developers, three overarching themes emerged: Productivity, Cognitive Effects, and Challenges. Each theme is organized into sub-themes, with illustrations of thematic maps and includes quotes from the interviews.

Table 2. Phases of Thematic Analysis and Their Descriptions

Phase	Description of the Process
1. Familiarizing yourself with your data	Transcribing data (if necessary), reading and re-reading the data, noting down initial ideas.
2. Generating initial codes	Coding interesting features of the data in a systematic fashion across the entire data set, collating data relevant to each code.
3. Searching for themes	Collating codes into potential themes, gathering all data relevant to each potential theme.
4. Reviewing themes	Checking if the themes work in relation to the coded extracts (Level 1) and the entire data set (Level 2), generating a thematic 'map' of the analysis.
5. Defining and naming themes	Ongoing analysis to refine the specifics of each theme, and the overall story the analysis tells, generating clear definitions and names for each theme.
6. Producing the report	The final opportunity for analysis. Selection of vivid, compelling extract examples, final analysis of selected extracts, relating back the analysis to the research question and literature, producing a scholarly report of the analysis.

4.1 Productivity

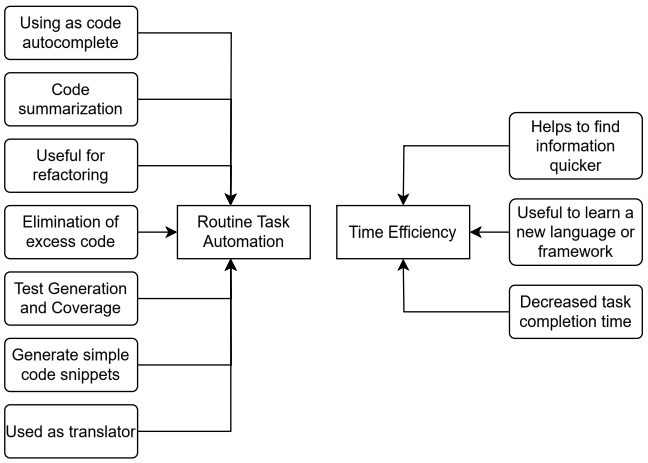

Fig. 1. Thematic Map for Theme 1: Productivity

Software developers aim to increase productivity, reduce task completion time to satisfy the clients and stakeholders: *"[...] in software engineering you should finish the job fast to deliver to customers."* (Participant 1). Therefore, productivity is crucial for software developers.

4.1.1 Routine task automation. Software developers' main use case is automating routine tasks, that are performed regularly in their workflow using LLM. When certain problems do not require human oversight and could be performed through monotonous work, software developers tend to use the LLM assistant for that: *"I often turn to LLM when the upcoming work is monotonous and voluminous, for example, converting an existing list of data in one format to another[...]"* (Participant 2). The other use scenario is translating data from one language to another, this could be the case for either programming or human language: *"The second case is the translation of texts in the application, content. When you need to translate a text into 10+ languages, it greatly speeds up the task and simplifies this routine.[...] it was necessary to transfer functionality from Android to iOS (from Kotlin to Swift), It's easier to ask to write or convert the code[...]"*(Participant 2). Developers often have to interact with the code of their colleagues, in order to understand the code they use LLMs to break down and refactor it.

4.1.2 Time efficiency. Participants noted that integrating LLMs in their workflow decreased the time they take to do their work: *"[...] now I do it all faster in Chat GPT. So productivity has increased several times."* (Participant 3). Software development industry relies on constant learning and adapting to the new environment, that is, being able to learn on the go. Rather than trawling through documentation or Stack Overflow, engineers could pose a natural-language question and receive an immediate, example-laden answer. It was mentioned that learning through LLM assistant is the main approach: *"I turn to LLM when I don't even know an approximate answer in advance, for example, I don't know what functionality is available."*(Participant 2).

4.2 Cognitive Impact

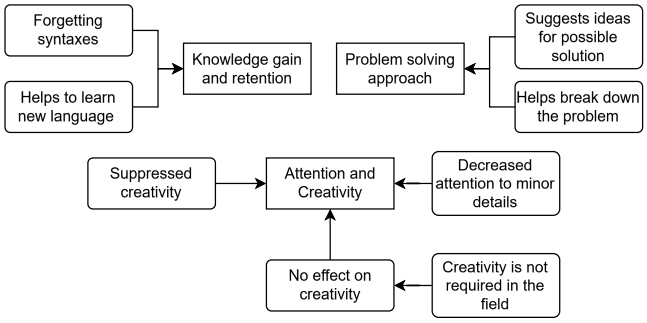

Fig. 2. Thematic Map for Theme 2: Cognitive Effects

The integration of LLMs into the software development workflow has not only improved productivity but also introduced cognitive shifts among developers. These changes are particularly evident in the areas of knowledge retention, attention to detail, and approaches to problem solving. While LLMs provide immediate assistance, they also influence how and what developers learn, raising concerns about long-term knowledge retention and critical thinking.

4.2.1 Knowledge gain and retention. One of the primary cognitive impacts reported by participants is the shift in how they acquire and retain technical knowledge. Developers frequently rely on LLMs to recall programming syntax or to gain familiarity with a new language or framework. Instead of memorizing commands or consulting official documentation, developers reported using LLMs to retrieve code snippets on demand. This approach enables immediate progress in coding tasks but potentially reduces long-term retention of knowledge: *"Yes, I started to turn to ChatGPT and Copilot more often to quickly find solutions or examples. It's convenient, but I understand that there is a risk of "forgetting how to think""*(Participant 4).

4.2.2 Attention and creativity. Another reported cognitive shift that concerns developers' attention to detail and creative engagement with their tasks. While LLMs help complete routine elements more efficiently, some participants observed a reduction in their attention toward minor details: *"I began to notice that I began to pay little attention to small syntaxes since the advent of LLM, sometimes I forget. Yes, this happens often now"* (Participant 5)

Moreover, the use of LLMs sparked contrasting views about creativity. Some participants felt that LLM suggestions could discourage original thinking: *Rather, it has influenced creativity, since I know that it is easier and faster to turn to Copilot than to waste time and think for myself.* (Participant 4). On the other hand, some participants either do not use the LLMs for creative tasks or do not think that the usage of LLM assistant impacted their creativity: *"Regarding the creativity, I think that it didn't make me uncreative."* (Participant 1).

4.2.3 Problem solving approach. LLMs also influence the way developers engage with problems. Some participants report that the tool serves as a cognitive assistance for deconstructing complex tasks into manageable steps: *"I usually use Copilot's suggestions as a draft. For example, when refactoring a large function, it suggests breaking it down into logical blocks — this helps structure the code and make it easier to maintain."* (Participant 4). Instead of jumping straight into coding, developers often begin by framing the problem in a prompt, using the LLM's response to get an idea for possible solutions. This iterative process allows them to clarify their approach before implementation. In contrast, some developers report that it did not affect how they approach certain problems: *" It didn't affect me at all how I approach tasks"* (Participant 5). The reason might be that they often try to exercise their own skills or do not prefer involving LLMs in their problem-solving process.

4.3 Challenges

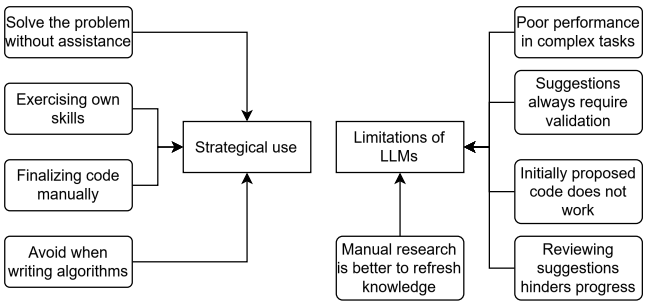

Fig. 3. Thematic Map for Theme 3: Challenges

Despite the productivity gains and cognitive shifts associated with LLM use, developers described a range of challenges and ways of finding a balance between productivity and cognitive effects. These challenges are represented by two intertwined dynamics: the need for strategic deployment of LLMs to protect problem-solving and independence, and the limitations of current models that can disrupt rather than accelerate work.

4.3.1 Strategic Use. Many participants mentioned that they try to solve problems on their own before turning to LLMs. They prefer to rely on their own knowledge first, especially when dealing with tasks that are more complex. Some even set time aside to work without LLMs, as a way to keep their skills sharp: *"I try to write tests by hand, because LLM can skip some test cases and plus it is useful to write tests yourself because this way more business logic remains in your head, all corner cases and you will remember and understand the project as a whole longer."* (Participant 3

Even when developers do use LLMs, they usually refrain copying the output directly. Instead, they review and improve the code themselves to make sure it works correctly, This shows that while LLMs help with problem solving, developers still want to stay in control of the final product: *"I always check and refine the final logic myself."* (Participant 4)

4.3.2 Limitations of LLMs. Integration of LLMs into software developers' workflow introduced some limitations in range and depth of tasks that it could be used for. Developers mentioned several instances where code suggestions by LLM did not perform as intended or could not be executed at all. While automating many processes in software development, LLMs have shortcomings that makes developers prefer manual coding and erodes trust in the LLM's ability to handle anything beyond trivial cases: *"[...] the written code is too superficial, often with errors, so that the project does not launch [...]"* Moreover, the necessity to validate every suggestion might hinder the progress, searching and correcting mistakes of the LLM can consume more time than writing the code from scratch. When initial suggestion is flawed, the iterative prompting can stall progress, reminding developers to remain autonomous and use the suggestions with caution .

4.4 Summary

Productivity was the first theme that emerged after grouping the codes. Developers reported that LLMs help automate repetitive or low-level coding tasks, such as translating, formatting, auto completing the code, or generate simple code snippets. These tasks, once time-consuming, were automated through simple prompts, allowing engineers to shift focus to high-level tasks. The use of LLMs allowed participants to approach problems more iteratively and efficiently. Overall, each of these factors contributed to positive improvements in perceived productivity. However, these productivity improvements also implied cognitive shifts in software developers. Developers described using LLMs to recall syntax, suggest solutions, or breaking down complex problems, which made them rely on the LLM more often. Several participants raised concerns about "forgetting how to think" or having problems with recall of exact syntax. Creativity was another topic that was concerned: some developers felt that LLM use limited their original thinking, while others believed that creativity was not important for most development tasks. Attention to detail also seemed to decline in some cases, as reliance on LLM-generated code reduced the urgency to manually check for small syntactical or logical errors. In response to these shifts, participants described methods they use to maintain their autonomy. Some emphasized the importance of using LLMs selectively: either as a guidance when tackling a problem or strictly for simple tasks. Final validation and analysis of the suggestion by the LLMs is still crucial. Participants are aware of the limitations of LLMs, especially in complex coding projects, where incorrect or incomplete responses required debugging or extensive prompting to achieve somewhat working prototype.

5 DISCUSSION

LLMs such as GitHub Copilot and ChatGPT have shown to be enhancing developer perceived productivity [21, 22]. This was strongly reflected in the theme of Productivity, where participants emphasized that LLMs accelerated development by automating routine tasks. The perceived productivity improvements are in line with findings on time savings and developer satisfaction when using AI programming assistants [22].

The theme of Cognitive Effects revealed contrasting experience: while LLMs help with tackling the problem, they often change the way developers approach the problem and affect long-term knowledge retention. This aligns with concerns raised in previous studies on cognitive offloading, where reliance on external tools can reduce application and retention of skills [18]. Moreover, several participants admitted to forgetting syntax or refraining from their previous problem solving approach and turning to LLMs for guidance.

This tension also appeared in participants' self-reflections on creativity and attention. Some reported feeling less creatively engaged, as LLMs offered ready-made solutions that reduced the need for creative thinking, supporting the arguments made in related studies that question whether AI tools suppress exploratory behavior in software tasks[4]. However, others suggested that creativity is not necessarily harmed by LLM use, revealing the fact in how creativity is perceived in software development. This insight shows that the relationship between creativity, problem solving, and AI support is not uniform, which was not stated in prior literature.

The theme of Challenges further highlighted the intentional, strategic ways developers manage their use of LLMs. Participants described conscious efforts to retain the knowledge, such as solving problems without assistance, manually finishing the logic. This mirrors concerns from prior studies, that human autonomy in problem solving must remain central even in AI-augmented workflows [13, 19].

Importantly, this study identified that participants acknowledge the cognitive effects caused by LLMs, however in order to increase the productivity, they attempt to prevent these effects by using methods that assist with coping and integrating LLMs in their workflow to minimize the negative cognitive impact. Developers frequently discussed the need to validate, rework, or even discard LLM-generated outputs when they did not meet expectations. This

cautious approach aligns with findings from related studies, who stress the importance of validation and user oversight in AI-driven code generation [14, 16].

6 LIMITATIONS AND FUTURE WORK

6.1 Limitations

While this study provides valuable qualitative insights into how software developers perceive the impact of LLM-based copilots on their problem-solving skills and independence, several limitations must be acknowledged. The sample size of 5 participants was relatively small, while suitable for exploratory research, but limits the depth of the findings. In addition, participants were selected through purposive sampling, potentially introducing selection bias. This study is based entirely on self-reported data through semi-structured interviews. While participants shared important insights, their responses may have been influenced by personal bias. It is also important to note that the interviews captured perceptions at a single point in time, without the ability to observe how these perceptions change long-term with LLM use.

6.2 Future Work

Future research should address these limitations by primarily expanding the sample size, and maintaining diversity in level of experience and industry. In addition, future work could utilize mixed methods, combining qualitative interviews with controlled experiments, to find connections between self-reported insights and behavioral patterns. This could offer a more objective understanding of the impact of LLMs on software engineering practices.

7 CONCLUSION

This study explored how software developers perceive the impact of large language models (LLMs) on their problem-solving skills and professional independence. By conducting semi-structured interviews and analyzing the responses through Braun and Clarke's thematic analysis framework, three key themes were identified: productivity, cognitive effects, and challenges.

The findings show that developers see LLMs as valuable tools for increasing productivity. However, developers also expressed caution about relying too heavily on LLMs, particularly in complex or critical tasks. Many participants noted a decrease in attention to small details and raised concerns about losing precise knowledge of programming languages, due to offloading auto completion to LLM assistant.

Developers strategically regulate their use of LLMs, often choosing to solve problems independently and validate the outputs of the models thoroughly. This behavior suggests that while LLMs are integrated into daily workflows, human oversight and critical thinking remain essential. Developers treat LLMs as assistants that can guide them or accelerate problem-solving, but not as replacements for their technical judgment or creative reasoning.

Answering the research question: How do software engineers perceive the impact of large language models on their problem-solving skills and independence? The developers acknowledge the impact on their problem-solving and independence. The usage of LLMs introduced over-reliance, effects on problem solving approach and knowledge retention. They are aware of the effects and risks, however, they are adapting strategies and methods to find a balance between increased productivity and cognitive effects.

REFERENCES

[1] Omolola A. Adeoye-Olatunde and Nicole L. Olenik. 2021. Research and scholarly methods: Semi-structured interviews. *JACCP: JOURNAL OF THE AMERICAN COLLEGE OF CLINICAL PHARMACY* 4, 10 (Oct. 2021), 1358–1367. https://doi.org/10.1002/jac5.1441

[2] Leonardo Banh, Florian Holldack, and Gero Strobel. 2025. Copiloting the future: How generative AI transforms Software Engineering. *Information and Software Technology* 183 (July 2025), 107751. https://doi.org/10.1016/j.infsof.2025.107751

[3] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (Jan. 2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

[4] Sergio Cavalcante, Erick Ribeiro, and Ana Oran. 2025. The Impact of AI Tools on Software Development: A Case Study with GitHub Copilot and Other AI Assistants:. In *Proceedings of the 27th International Conference on Enterprise Information Systems*. SCITEPRESS - Science and Technology Publications, Porto, Portugal, 245–252. https://doi.org/10.5220/0013294700003929

[5] Anita N. Chary, Noelle Castilla-Ojo, Christopher Joshi, Ilianna Santangelo, Christopher R. Carpenter, Kei Ouchi, Aanand D. Naik, Shan W. Liu, and Maura Kennedy. 2022. Evaluating older adults with cognitive dysfunction: A qualitative study with emergency clinicians. *Journal of the American Geriatrics Society* 70, 2 (Feb. 2022), 341–351. https://doi.org/10.1111/jgs.17581

[6] Joel Cordeiro, Bruno Antunes, and Paulo Gomes. 2012. Context-based recommendation to support problem solving in software development. In *2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE)*. IEEE, Zurich, Switzerland, 85–89. https://doi.org/10.1109/RSSE.2012.6233418

[7] Thomas J. D'Zurilla and Marvin R. Goldfried. 1971. Problem solving and behavior modification. *Journal of Abnormal Psychology* 78, 1 (Aug. 1971), 107–126. https://doi.org/10.1037/h0031360

[8] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, Melbourne, Australia, 31–53. https://doi.org/10.1109/ICSE-FoSE59343.2023.00008

[9] Nicole Forsgren, Margaret-Anne Storey, Chandra Maddila, Thomas Zimmermann, Brian Houck, and Jenna Butler. 2021. The SPACE of Developer Productivity: There's more to it than you think. *Queue* 19, 1 (March 2021), 20–48. https://doi.org/10.1145/3454122.3454124

[10] Michael Gerlich. 2025. AI Tools in Society: Impacts on Cognitive Offloading and the Future of Critical Thinking. *Societies* 15, 1 (Jan. 2025), 6. https://doi.org/10.3390/soc15010006

[11] J. Paul Gibson and Jackie O'Kelly. 2005. Software engineering as a model of understanding for learning and problem solving. In *Proceedings of the 2005 international workshop on Computing education research - ICER '05*. ACM Press, Seattle, WA, USA, 87–97. https://doi.org/10.1145/1089786.1089795

[12] Greg Guest, Emily E. Namey, and Marilyn L. Mitchell. 2013. *Collecting Qualitative Data: A Field Manual for Applied Research*. SAGE Publications, Ltd, 1 Oliver's Yard, 55 City Road London EC1Y 1SP. https://doi.org/10.4135/9781506374680

[13] Ranim Khojah, Mazen Mohamad, Philipp Leitner, and Francisco Gomes De Oliveira Neto. 2024. Beyond Code Generation: An Observational Study of ChatGPT Usage in Software Engineering Practice. *Proceedings of the ACM on Software Engineering* 1, FSE (July 2024), 1819–1840. https://doi.org/10.1145/3660788

[14] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ACM, Lisbon Portugal, 1–13. https://doi.org/10.1145/3597503.3608128

[15] James Mcguire. 2001. What is problem solving? A review of theory, research and applications. *Criminal Behaviour and Mental Health* 11, 4 (Nov. 2001), 210–235. https://doi.org/10.1002/cbm.397

[16] Ipek Ozkaya. 2023. Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Software* 40, 3 (May 2023), 4–8. https://doi.org/10.1109/MS.2023.3248401

[17] Jude Page, Timothy Broady, Sujith Kumar, and Evelyne De Leeuw. 2022. Exploratory Visuals and Text in Qualitative Research Interviews: How Do We Respond? *International Journal of Qualitative Methods* 21 (April 2022), 16094069221110302. https://doi.org/10.1177/16094069221110302

[18] Evan F. Risko and Sam J. Gilbert. 2016. Cognitive Offloading. *Trends in Cognitive Sciences* 20, 9 (2016), 676–688. https://doi.org/10.1016/j.tics.2016.07.002

[19] P.N. Robillard. 2005. Opportunistic Problem Solving in Software Engineering. *IEEE Software* 22, 6 (Nov. 2005), 60–67. https://doi.org/10.1109/MS.2005.161

[20] Zibin Zheng, Kaiwen Ning, Qingyuan Zhong, Jiachi Chen, Wenqing Chen, Lianghong Guo, Weicheng Wang, and Yanlin Wang. 2025. Towards an understanding of large language models in software engineering tasks. *Empirical Software Engineering* 30, 2 (March 2025), 50. https://doi.org/10.1007/s10664-024-10602-0

[21] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. ACM, San Diego CA USA, 21–29. https://doi.org/10.1145/3520312.3534864

[22] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2024. Measuring GitHub Copilot's Impact on Productivity. *Commun. ACM* 67, 3 (March 2024), 54–63. https://doi.org/10.1145/3633453