DOI: https://doi.org/10.4038/jdrra.v2i1.57

The Journal of Desk Research Review and Analysis, Vol. 2, Issue 1, 2024, 235-249

SYSTEMATIC LITERATURE REVIEW ON ANALYSING THE IMPACT OF PROMPT ENGINEERING ON EFFICIENCY, CODE QUALITY, AND SECURITY IN CRUD APPLICATION DEVELOPMENT

KAA Shanuka[1], J Wijayanayake[2] and K Vidanage[3]

Abstract

This research investigates the impact of prompt engineering on the efficiency, code quality, and security of CRUD (Create, Read, Update, Delete) operations in software development using large language models (LLMs) like ChatGPT. Prompt engineering, which involves crafting specific inputs to guide AI outputs, has become crucial for code generation, debugging, and vulnerability detection tasks. The study addresses three key research questions: identifying the most influential aspects of prompt engineering on CRUD efficiency compared to traditional coding, recognising common error patterns in AI-generated code and preemptive mitigation strategies, and leveraging human-AI collaboration to optimise CRUD application development. This study evaluates 52 relevant papers from 2018 to 2023 through a systematic literature review from Google Scholar, ResearchGate, arXiv, and Sci-Hub. The findings indicate that effective, prompt engineering significantly enhances productivity, reduces development time, and improves code adaptability to complex data structures. Techniques such as role-prompting and chain-of-thought prompting are highlighted for their ability to produce high-quality code, reducing errors and enhancing overall code quality. The review revealed significant results, including identifying prompt engineering techniques that substantially improve the performance of LLMs in CRUD operations. ChatGPT, in particular, excels in generating and debugging code but struggles with more complex tasks and security vulnerabilities. The study emphasises the need for continuous human oversight to ensure security and correctness, addressing challenges such as managing programming variability and the stochastic nature of LLMs.The research concludes that integrating ChatGPT into CRUD operations significantly advances software development, improving efficiency, code quality, and security. However, it is crucial to address its limitations through better prompt engineering and continuous human involvement. Future work will involve developing a fully functional CRUD application using generative AI, further evaluating its practical applications and limitations in software development.

Keywords: ChatGPT, Code Quality, CRUD Operations, Large Language Models, Programming, Prompt Engineering

[1] Department of Industrial Management, Faculty of Science, University of Kelaniya

Email: ashenshanuka1222@gmail.com https://orcid.org/0009-0006-6940-817X

[2] Professor, Department of Industrial Management, University of Kelaniya,

Email: janaka@kln.ac.lk https://orcid.org/0000-0002-9523-5384

[3] Senior Lecturer, Department of Computer Science, General Sir John Kotelawala Defence University

Email: vidanage_bvki@kdu.ac.lk https://orcid.org/0000-0002-1923-9084

Received date: 20.08.2024
Accepted date: 22.10.2024

Introduction

Prompt engineering has become crucial in software development with large language models (LLMs) like ChatGPT. This technique involves crafting specific inputs to guide LLM outputs, enhancing code generation, debugging, and security. Generative AI tools like ChatGPT, GitHub Copilot, Gemini, and Code Whisperer excel in tasks like code summarisation, bug triaging, and program repair. CRUD operations—Create, Read, Update, and Delete—are fundamental for managing data in software applications. Advanced prompt engineering techniques have revolutionised CRUD operations, automating repetitive tasks, reducing development time, and improving code quality and adaptability to complex data structures using generative AI platforms. By the end of this research, the following questions aim to be answered,

> RQ 1: What specific aspects of prompt engineering most significantly influence the efficiency of CRUD operations when using generative AI models, and how do these influences compare with traditional manual coding approaches?
> RQ 2: What are the common patterns of errors or vulnerabilities in AI-generated code for CRUD operations, and how can these be preemptively identified and mitigated?
> RQ 3: How can human-AI collaboration be leveraged to enhance the development process of CRUD applications, and what are the roles and responsibilities of human developers and AI models in this collaborative approach?

By the end of this research, the following objectives are anticipated to be achieved,

1. To identify critical factors influencing the efficiency of CRUD operations using generative AI.
2. To analyse existing large language models for code quality and security in CRUD applications.
3. To compare the performance of AI-generated CRUD applications against traditional coding practices based on findings from the literature.
4. To assess the reported outcomes of human-AI collaboration in CRUD application development through systematic analysis of existing studies.
5.

The effectiveness of AI-generated code varies, with ChatGPT excelling at simpler tasks but struggling with complex challenges and security issues.(Kuhail et al., 2024) Prompt engineering enhances CRUD application development's efficiency, code quality, and security. This research aims to analyse prompt engineering's impact, identify key factors, and evaluate human-AI collaboration, providing insights to optimise AI integration in software development.

Methodology

The primary objective of this study is to review existing literature on CRUD application development using generative AI, focusing on the impact of prompt engineering on efficiency, code quality, and security. Following PRISMA guidelines(Page et al., 2021; *PRISMA Statement*, n.d.) for a systematic and transparent review, relevant papers were sourced from Google Scholar, ResearchGate, arXiv, and sci-Hub. The initial dataset comprised 7,410 records, refined by applying specific inclusion and exclusion criteria.

Initially, 857 records were excluded for not using surveys or questionnaires for data collection. An additional 385 records were removed because quantitative or mixed research methodologies were lacking. Only the first forty pages of Google Scholar results were analysed to keep the dataset manageable. Papers irrelevant to the research, unavailable in full, or not published in English were excluded. Only papers from 2018 to 2023 were considered, resulting in 52 papers selected for final evaluation. (Fig.1)

The review included detailed searches using specific keywords related to the research topic, such as "ChatGPT in software engineering," "Prompt engineering in programming", "AI in software development," "Security in AI-generated code", "Code quality metrics in AI programming" "ChatGPT" AND "CRUD" "CRUD" AND "ChatGPT" These searches were conducted across multiple databases.

The selected papers were analysed for the role of prompt engineering in enhancing large language models (LLMs) for CRUD operations, code generation accuracy, productivity, and the effectiveness of AI tools like ChatGPT in debugging and improving code quality. The study also examined the security implications of AI-generated code and how prompt engineering can mitigate vulnerabilities. The synthesised data provides insights into the practical applications and limitations of integrating AI tools in software development, especially for CRUD applications.

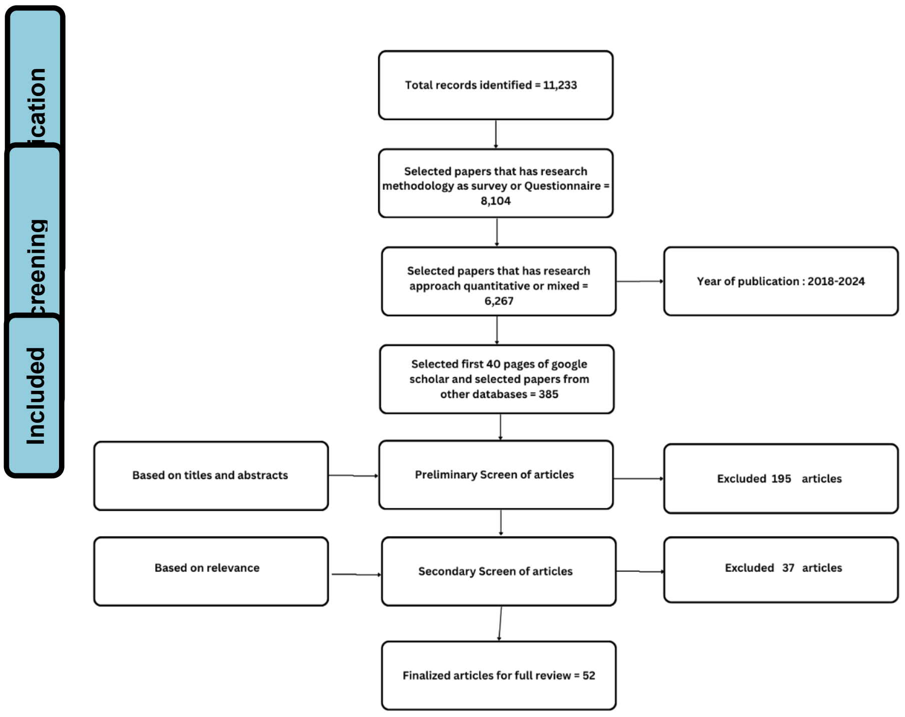
cation
creening
Included
Total records identified = 11,233
Selected papers that has research methodology as survey or Questionnaire = 8,104
Selected papers that has research approach quantitative or mixed = 6,267
Year of publication : 2018-2024
Selected first 40 pages of google scholar and selected papers from other databases = 385
Based on titles and abstracts
Preliminary Screen of articles
Excluded 195 articles
Based on relevance
Secondary Screen of articles
Excluded 37 articles
Finalized articles for full review = 52

Fig. 1: PRISMA Flow Diagram

Keywords
AI, ChatGPT, Coding, CRUD operations, Generative AI, Programming, Software Engineering

Results and Discussion

Prompt Engineering on Software Development

Prompt engineering involves crafting specific text prompts that generative AI models can understand and act upon effectively. This technique is crucial for enhancing the efficiency of large language models (LLMs) by providing clear, specific, and unambiguous instructions. Effective prompts include sufficient context, detailed instructions, and examples, often requiring iterative refinement to achieve accuracy and alignment with the intended goal(Hemil et al., 2024; Jiang et al., 2022; Sahoo et al., 2024).

Prompt engineering optimises LLM performance in software development, making it a cornerstone of current AI applications(Chen et al., n.d.; 'Exploring the Competency of ChatGPT in Solving Competitive Programming Challenges', 2023). It has evolved into an advanced industry with methods ranging from foundational approaches like role-prompting to advanced techniques such as chain-of-thought and tree-of-thought prompting. These methods reduce machine hallucinations by integrating external knowledge, improving the accuracy of LLM outputs(Chen et al., n.d.). Generative AI, through prompt engineering, has the potential to revolutionise end-user programming by enabling code generation from natural language prompts, significantly expanding its scope(Sarkar, 2023).

LLMs such as Copilot, CodeWhisperer, and ChatGPT have demonstrated exceptional performance in various software engineering tasks, including code summarisation, bug triaging, and program repair(Dipongkor, 2024; Weisz et al., 2023; Yetiştiren et al., 2023), (Table 1).

	Copilot v1.70.8099 (New)			**CodeWhisperer Jan '23 (New)**			**ChatGPT 9 Jan '23 Version**		
	ORG	OFN	DFN	ORG	OFN	DFN	ORG	OFN	DFN

Table 4: Percentage Results of all code generation tools for Original Experiment (ORG), Only Function Name (OFN) and Dummy Function Name (DFN)

Source: **(Yetiştiren et al., 2023)**

Effective prompt creation, like the CEDAR technique, significantly improves accuracy and efficiency in tasks such as test assertion generation and program repair, outperforming existing models(Nashid et al., 2023). LLMs enhance code quality by reducing complexity, improving readability, and generating correct and less complex programs(Shirafuji et al., 2023).

Despite the benefits, prompt engineering faces challenges such as managing programming variability and the stochastic nature of LLMs(Acher et al., 2023; Rush, 2023). Leveraging few-shot examples with LLMs like GPT-3.5 promotes better programming practices, though some unnecessary modifications may occur(Shirafuji et al., 2023). The interaction between humans and generative AI tools in creative industries reveals diverse strategies to overcome these challenges, emphasising the evolving nature of these technologies(Feng et al., 2024; Rush, 2023; Sun et al., 2024).

Prompt engineering is essential for optimising LLMs in software development, significantly enhancing efficiency, code quality, and security. Techniques like role-prompting, chain-of-thought prompting, and tree-of-thought prompting have proven effective in reducing machine hallucinations and improving output accuracy. Despite challenges, prompt engineering remains crucial for advancing LLM capabilities, driving innovation, and improving productivity in software development (Huang et al., 2024; Ricárdez et al., 2024; Taeb et al., 2024).

ChatGPT as a Software Development Tool

ChatGPT is a helpful software development tool known for generating code, debugging, and enhancing security. Its advanced language skills allow it to create accurate code, find and fix errors, and spot vulnerabilities. It boosts productivity and efficiency but has limitations, such as struggling with complex errors and sometimes giving lengthy responses. ChatGPT is best used as a support tool that complements human expertise, helping ensure strong and secure software development.

Code Generation Accuracy and Productivity of ChatGPT

OpenAI has transformed ChatGPT from its initial versions to the sophisticated GPT-4, significantly enhancing its programming capabilities. Built on a transformer-based architecture, ChatGPT excels in tasks like code generation and debugging, making it one of the most accurate AI tools in software development(Coello et al., 2024) (Georgsen, n.d.). Since its release, developers have widely adopted ChatGPT for its efficiency and reliability in software engineering, supporting them across various development lifecycle stages. This introduction explores the evolution, structure, and impactful role of ChatGPT in modern software development.

ChatGPT has demonstrated a mixed performance in terms of code generation accuracy, especially when tested on diverse coding problems from platforms like LeetCode. While it excels at solving easier and more popular coding problems, it struggles significantly with harder and less popular ones(Kuhail et al., 2024). For instance, ChatGPT solved 73.3% of easy problems but only 10% of complex problems. This disparity highlights the tool's limitations in handling complex coding tasks efficiently(Kuhail et al., 2024). Similarly, Codex, another AI tool with 12 billion model parameters trained on 54 million GitHub repositories, showed a 70% success rate on Python programming problems, indicating that while AI tools are promising, their effectiveness diminishes with problem complexity. (Chen et al., n.d.)

In a study comparing the performance of ChatGPT with Stack Overflow for various programming tasks, ChatGPT outperformed Stack Overflow in algorithmic and library tasks but lagged in debugging tasks. This suggests that while ChatGPT can generate high-quality code and complete tasks quickly, its utility varies depending on the task type (J. Liu et al., 2023). Furthermore, a user study involving 99 programmers revealed that frequent users of AI tools tend to trust them more. (Ahmad et al., 2023; Wermelinger, 2023)

AI tools like ChatGPT and GitHub Copilot have significantly boosted productivity. For example, GitHub Copilot's autocomplete functionality has been associated with a 50% increase in productivity(Wermelinger, 2023). A study by (Noy & Zhang, 2023)involving 444 participants found that ChatGPT-3 reduced coding time by 0.8 standard deviations and improved output quality by 0.4 standard deviations. However, ChatGPT's performance is not without flaws. It can sometimes provide incorrect or overly verbose responses due to its optimisation tendencies and limited inference capabilities, making it less reliable in certain contexts (Sarkar, 2023).

ChatGPT, particularly in its advanced GPT-4 iteration, has markedly enhanced code generation accuracy and productivity in software development. It is highly effective in generating and debugging code, making it a valuable tool for developers across various development lifecycle stages. While it excels in solving more straightforward coding tasks and improving efficiency, its performance diminishes with more complex problems. Despite its strengths, ChatGPT can sometimes produce incorrect or overly verbose outputs. ChatGPT has proven to be a significant asset in software engineering, streamlining processes and increasing productivity, though there remains room for improvement in handling more complex tasks.

Debugging Capabilities and Limitations of ChatGPT

ChatGPT, an advanced natural language processing (NLP) model, has gained significant attention for its ability to assist developers in various tasks, including code debugging. Its debugging capabilities leverage NLP skills to analyse and identify errors in code, regardless of the programming language(Haque & Li, 2023). ChatGPT offers several advantages as a debugging tool, such as increased efficiency, improved code quality, continuous learning from previous sessions, and the potential for offering debugging as a service. However, its limitations include a limited understanding of complex errors, dependence on training data, and a tendency to generate overly verbose responses(Haque & Li, 2023). Fig. 2 and 3 illustrate the comparison between traditional approaches and the debugging process of ChatGPT.(Haque & Li, 2023)

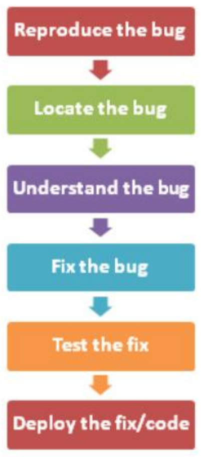

Fig. 2: Typical Debugging Process using Traditional Methods

***Source:* (Haque and Li, 2023)**

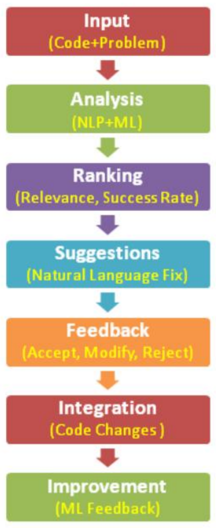

Fig. 3 Debugging Process Using ChatGPT

***Source:* (Haque and Li, 2023)**

Despite these drawbacks, studies have shown ChatGPT to be effective in identifying syntax errors and common mistakes, suggesting code optimisations, refactoring, and generating missing code based on project context(Biswas, 2023). Nonetheless, ChatGPT's ability to detect vulnerabilities is inconsistent, often making erroneous predictions when verifying the correctness of generated code. This highlights the need for more effective prompt design to enhance its self-verification capabilities(Yu et al., 2024).

ChatGPT should be used as an augmentative tool rather than a replacement for human expertise in debugging. While it can automate certain aspects of debugging, saving time and effort, human review and testing remain critical components of the software development process(Haque & Li, 2023). Thus, ChatGPT's role in debugging is to complement human efforts, leveraging its strengths to improve efficiency while acknowledging its limitations in addressing complex errors and maintaining real-time interaction(Biswas, 2023; Haque & Li, 2023; Yu et al., 2024).

Enhancing Security in Software Development of ChatGPT

The role of ChatGPT in enhancing security within software development is multifaceted. It can assist in detecting vulnerabilities by analysing code and comparing it to its training data, but its effectiveness often depends on the quality of the prompts provided(Zhang et al., 2024).

A study on using ChatGPT for vulnerability detection found that while ChatGPT can detect vulnerabilities, it often provides incorrect predictions when explicitly asked if the completed code is free of vulnerabilities. This is due to its limited ability to self-verify and the occurrence of self-contradictory hallucinations, where it initially generates what it believes to be correct code but later predicts it to be incorrect during self-verification(Zhang et al., 2024). The security implications of using AI tools like ChatGPT are significant. Developers need to be cautious about trusting AI-generated code due to the risk of insecure code propagation. A study comparing ChatGPT and Stack Overflow found that while ChatGPT generated fewer vulnerabilities overall, the differences were not statistically significant, indicating that both platforms can propagate vulnerable code(Hamer et al., 2024). The study also emphasised the importance of applying good software security practices, such as static analysis tools and software testing, to detect and mitigate vulnerabilities in AI-generated code (Hamer et al., 2024).

For instance, adding high-quality code summaries to prompts can improve ChatGPT's performance in vulnerability detection, although the impact is programming-language-specific (Zhang et al., 2024)

ChatGPT plays a multifaceted role in enhancing security within software development by assisting in vulnerability detection through code analysis. However, its effectiveness is highly dependent on prompt quality. Studies have shown that while ChatGPT can identify vulnerabilities, it often provides incorrect predictions due to limited self-verification abilities and occasional hallucinations. Therefore, developers should not blindly trust AI-generated code and must employ reasonable security practices, such as static analysis tools and thorough software testing, to ensure code security. Improving prompt quality, like adding detailed code summaries, can enhance ChatGPT's vulnerability detection performance, though results may vary by programming language.

CRUD operations as Software development's backbone

CRUD operations, an acronym for Create, Read, Update, and Delete, are essential for managing data in software applications. These operations streamline database interactions, maintain data integrity, and enhance performance, making them crucial for developing robust and user-friendly applications(Bartalos & Bieliková, n.d.).

Different technologies showcase the strengths and limitations of CRUD. NoSQL databases, such as MongoDB and CouchDB, are praised for their flexibility and scalability. MongoDB excels in data fetching, whereas CouchDB performs well in write operations(Truica et al., 2015). On the other hand, relational databases using SQL offer robust data management but face challenges with scalability and performance in cloud-based environments(Bonteanu & Tudose, 2024). Comparative studies highlight MongoDB's speed in fetching data and CouchDB's efficiency in insert, update, and delete operations(Truica et al., 2015).

Frameworks like Spring Boot and tools like CRUDyLeaf automate these operations, enhancing development speed and reducing manual effort(Gómez et al., 2020). Despite these advancements, integrating features like authentication and search into CRUD-generated applications remains challenging(Anuar et al., 2023). Advanced patterns like CQRS and ES offer potential improvements by separating read and write operations but require deep domain understanding(Pantelelis & Kalloniatis, 2022).

HTTP CRUD methods (POST, PUT, DELETE) enhance API expressiveness, simplifying data management and supporting advanced querying(Schröder et al., 2018). Generative AI tools, such as ChatGPT, automate repetitive CRUD tasks, reducing development time, improving code quality, and adapting to complex data structures, thereby boosting productivity and robustness in software applications. As software complexity increases, leveraging AI to optimise CRUD operations will be crucial for maintaining effective, secure, and scalable applications.

Combining essential CRUD operations with modern Generative AI significantly enhances software development efficiency, scalability, and robustness.

Evaluation of AI-Generated Codes

Evaluating AI-generated code is crucial for understanding its practical applications and limitations. While AI boosts productivity and reduces development time, challenges in code correctness, maintainability, and security persist. This analysis focuses on the efficiency, quality, and security of AI-generated code, identifying strengths and areas for improvement.

Efficiency and Quality of AI-Generated Code

The efficiency and quality of AI-generated code from models like ChatGPT have gained significant attention in software development. OpenAI's advancements, particularly with GPT-4, have greatly enhanced ChatGPT's programming capabilities, making it a reliable tool for code generation and debugging (Sakib et al., 2023). (Y. Liu et al., 2024)However, its performance varies across different tasks, declining for new programming challenges introduced after January 2022(Y. Liu et al., 2024). The quality of the generated code also decreases with task complexity, as seen in Python (63% clean code for passed tasks vs. 56% for failed tasks) and Java (47% clean for passed vs. 39% for failed) (Y. Liu et al., 2024).

Technical evaluations reveal mixed results. While ChatGPT 3.5 produced accurate Python code for various computational problems, ChatGPT 4.0 encountered more runtime errors and inaccuracies (Diehl et al., 2024). It performed well in C++ and Java but struggled with complex and parallel computing constructs. Errors were rare in easy tasks (1%) but increased for more complex tasks (10%), with common issues including incorrect outputs (27%) and code style challenges (47%)(Y. Liu et al., 2024).

Despite its promise, ChatGPT faces challenges such as efficiency drops with increasing task complexity, overly verbose responses, and incorrect predictions due to limited self-verification (Haque & Li, 2023). Human review and testing remain crucial for handling complex errors and ensuring security (Yu et al., 2024). Improving prompt engineering, leveraging detailed feedback, and iterative repairing can enhance ChatGPT's performance, addressing its code generation and debugging limitations.

Security in AI-Generated Code

The security of AI-generated code is a paramount concern, particularly given the potential for introducing vulnerabilities during code generation. ChatGPT has shown some ability to detect and address software vulnerabilities, but its performance varies significantly depending on the type of vulnerability and the context provided(Khoury et al., 2023; Yin, 2024). For example, while ChatGPT can identify issues such as bound checking and input validation, it often overlooks other critical vulnerabilities like global variable misuse and format string vulnerabilities(Taeb et al., 2024).

A comparative study evaluating ChatGPT, CodeBert, and CodeX found that CodeX had the highest code generation capability, producing accurate, secure, and privacy-preserving code. In contrast, ChatGPT excelled in explaining potential vulnerabilities and providing detailed comments, which is beneficial for understanding security implications (Taeb et al., 2024). Fig.4 illustrates the overall performance of the ChatGPT, CodeBert, and CodeX models.
They discovered that ChatGPT's efficacy and security handling performance is significantly higher than other code generation models.

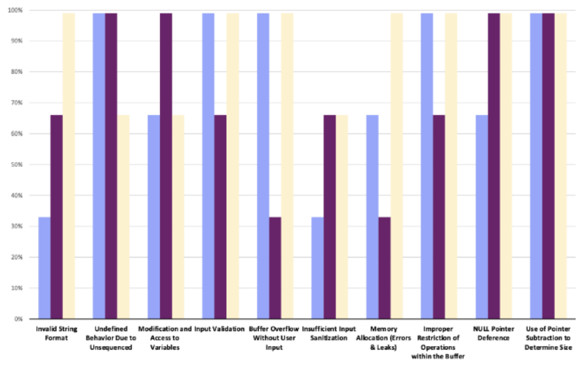

Fig. 4: Overall Performance of LLM Models
Source: (Taeb et al., 2024)

However, ChatGPT's ability to repair vulnerabilities is limited, regardless of whether context information is provided (Jaber et al., 2023; Yin, 2024).

Further analysis revealed that ChatGPT's vulnerability detection could be easily influenced, indicating a lack of confidence in its own detection capabilities. Moreover, its capacity to assess vulnerability severity based on source code alone is limited but can be improved with additional context (Yin, 2024). This highlights the need for integrating more comprehensive context information and advanced prompt engineering techniques to enhance the security evaluation capabilities of AI models like ChatGPT.

Despite these limitations, ChatGPT's role in improving security through prompt engineering shows potential. For instance, iterative prompt refinement and feedback from static analysis tools can help identify and mitigate security vulnerabilities more effectively (Y. Liu et al., 2024). Combined with continuous input from practitioners, this approach can improve the robustness of AI-generated code and enhance its security posture. (Botacin, 2023) investigates GPT-3's potential use in generating malware, revealing its capability to create malicious code through smaller building blocks despite challenges with large code chunks. While GPT-3 can aid in malware creation, it also assists in deobfuscating threats, highlighting its dual-purpose nature. The findings emphasise the need for complementary solutions to address these security concerns and the variable detection rates of generated malware variants.

The security of AI-generated code, particularly from ChatGPT, remains a critical concern. While ChatGPT shows promise in identifying and explaining vulnerabilities, its ability to repair them is limited and highly context-dependent. Comprehensive context integration and advanced prompt engineering techniques are essential to enhance its security capabilities. Iterative prompt refinement, feedback from static analysis tools, and continuous practitioner input can significantly improve the robustness and security of AI-generated code, addressing the potential risks of vulnerabilities and malicious code creation.

Human-AI collaboration in software development

Human-AI collaboration in software development is rapidly advancing, driven by tools like GitHub Copilot and ChatGPT, which enhance productivity and streamline coding tasks. GitHub Copilot acts as an "AI pair programmer," translating natural language descriptions into source code across multiple programming languages(Ahmad et al., 2023; Coello et al., 2024; Wermelinger, 2023). ChatGPT supports various aspects of software development, including programming, testing, and debugging. For example, it facilitated the development of an online behavioural task using HTML, CSS, and JavaScript, demonstrating its ability to script solutions and reduce developers' time and effort (Chauvet et al., n.d.).

The collaboration extends beyond code generation. ChatGPT assists in architecture-centric software development, synergising architects' knowledge with its capabilities to create robust systems(Ahmad et al., 2023). ChatGPT's automated bug-fixing performance enhances software testing processes(Sobania et al., 2023). Fig.5 illustrates the number of occurrences of identified classes of ChatGPT answers given for the problems from QuixBugs.

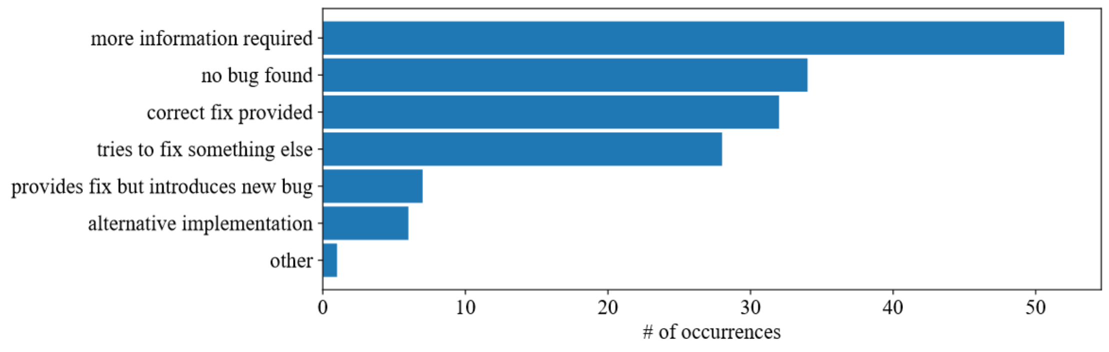

Fig. 5: Number of occurrences of identified classes of ChatGPT answers given for the problems from QuixBugs
***Source:* (Sobania et al., 2023)**

However, despite these advancements, concerns about AI potentially displacing human workers persist(Weidinger et al., 2021). AI tools should augment rather than replace human abilities, exemplified by GitHub Copilot handling low-level code details, allowing engineers to focus on higher-level design (Weidinger et al., 2021).

AI collaboration also enhances research by leveraging AI for data collection and emulating human responses, complementing traditional methods(Gerosa et al., 2024). For some coding problems, ChatGPT-4 successfully generated a working solution on the first attempt but struggled with more complex problems, requiring multiple iterations or failing to pass all test cases (Nascimento et al., 2023). ChatGPT outperformed novice programmers in easy and medium-level problems but could not outperform experienced programmers, highlighting the need for a collaborative approach (Nascimento et al., 2023).

Generative AI tools enhance developers' productivity by reducing the time spent on repetitive and mundane tasks. A study involving 444 participants revealed that ChatGPT significantly boosted productivity, reducing coding time by 0.8 standard deviations and improving output quality by 0.4 (Noy & Zhang, 2023). This productivity boost is attributed to the AI's ability to quickly produce satisfactory output, allowing developers to focus on more critical aspects of software development.

ChatGPT supports programming instruction in education by providing personalised and interactive practices, debugging assistance, and clear code explanations despite challenges such as inaccurate responses and ethical concerns(Husain, 2024). ChatGPT's programming skills need improvement; human expertise remains crucial in software engineering.(Koubaa et al., 2023)

Problem name	Difficulty	Laguage	Number of solutions that worked	If failed-number of test cases	Runtime (ms)	Memory (MB)	Performance (beats) (%)	Memory (beats) (%)	Time Complexity	Space Complexity	ID LeetCode submission
2656. Maximum Sum With Exactly K Elements	Easy	C++	1/1	-	47	70.6	70.22	68.69	O(n log n + k^2 log n)	O(n)	951082025
2656. Maximum Sum With Exactly K Elements	Easy	C++	2/3	-	62	71.7	21.75	12.19	O(n + k * log n)	O(n)	946957357
2657. Find the Prefix Common Array of Two Arrays	Medium	C++	1/1	-	53	80.9	78.35	97.47	O(n^3)	O(n)	951078174
2658. Maximum Number of Fish in a Grid	Medium	C++	1/1	-	84	88.4	69.14	88.14	O((mn)²)	O(mn)	951076414
2659. Make Array Empty	Hard	C++	0	505/514	-	-	-	-	O(n^2)	O(n)	951086982

Table 5: Result analysis of coding solutions provided by ChatGPT

***Source:* (Nascimento et al., 2023)**

Generative AI tools aid in code improvement and supplementary tasks, with 32.8% of generated code being unused due to quality concerns and only 16.8% integrating directly into master branches(Jin et al., 2024). Fig. 6 presents graphical representations of how ChatGPT impacts developers. (Vaillant et al., 2024)ChatGPT has been shown to produce correct code for a majority of tasks, with success rates varying based on problem difficulty. The integration of AI in software development offers substantial productivity gains but raises concerns about job security and the quality of AI-generated code(Husain, 2024; J. Liu et al., n.d.; Vaillant et al., 2024; Weidinger et al., 2021)

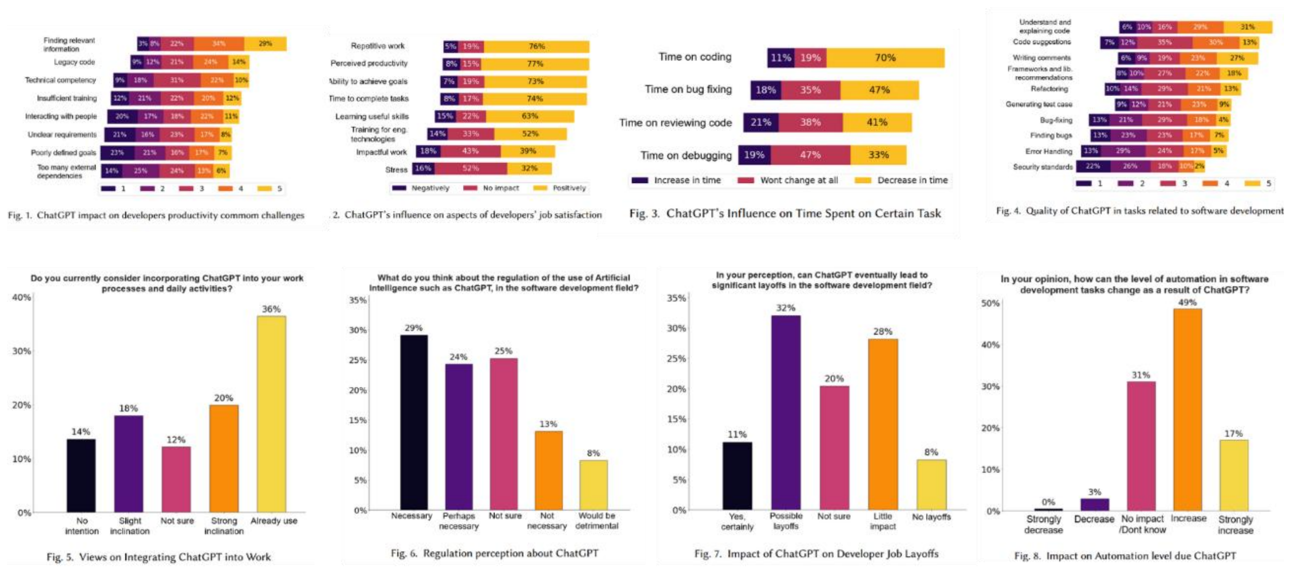

Fig. 6: Impact of CHATGPT as an automation tool on developers.
***Source:* (Vaillant et al., 2024)**

Overall, human-AI collaboration in software development highlights a symbiotic relationship where AI tools enhance human capabilities, improving efficiency and innovation. Future research should optimise this collaboration to maximise benefits while addressing associated challenges.

Conclusion

The integration of ChatGPT and other large language models (LLMs) into CRUD (Create, Read, Update, Delete) operations has significantly impacted software development by enhancing efficiency, code quality, and security. ChatGPT, through prompt engineering, involves creating detailed and specific instructions that guide the AI in performing tasks accurately. This process is essential for optimising the performance of LLMs, making them valuable tools in modern software engineering.

ChatGPT improves efficiency by automating repetitive coding tasks, thereby reducing development time and allowing developers to focus on more complex and creative aspects of their work. The AI's ability to generate functionally correct and clean code from natural language prompts showcases its potential to streamline development processes. Techniques like role-prompting and chain-of-thought prompting are employed to guide the AI in producing high-quality code that is both readable and maintainable. These techniques help in reducing errors and enhancing overall code quality.
Regarding security, ChatGPT can detect and address some vulnerabilities in the generated code. The AI can suggest fixes and improvements by analysing the code based on its training data, thus contributing to developing more secure software. However, the effectiveness of ChatGPT in identifying and fixing security issues depends heavily on the quality of the prompts and the context provided. Continuous human oversight is required to ensure that the code is secure and free from vulnerabilities.

Despite these advantages, ChatGPT has limitations. Its performance declines with more complex coding tasks, often producing overly verbose or incorrect code. The accuracy and usefulness of ChatGPT's outputs are heavily dependent on the quality of the prompts, and poorly crafted prompts can lead to suboptimal or incorrect code generation. Additionally, while ChatGPT can detect some security vulnerabilities, its ability to secure the code entirely is limited and necessitates human intervention.

In conclusion, the integration of ChatGPT into CRUD operations presents a significant advancement in software development, improving efficiency, code quality, and security. However, it is crucial to address its limitations through better prompt engineering and continuous human involvement. As technology evolves, refining these AI tools and effectively integrating them into development processes will be key to maximising their benefits and mitigating potential risks.

Future Works

Future work should focus on developing a fully functional CRUD application using generative AI prompts with a selected programming language and a large language model (LLM). This project should evaluate the application's performance, code quality, and security, providing insights into the practical applications and limitations of integrating generative AI in software development.

Acknowledgements
I would like to express my sincere gratitude to my supervisors, Prof. Janaka Wijayanayake, Dr. Kaneeka Vidanage, and Jinendri Prasadika, for their invaluable guidance and support throughout this research on "Analysing the Impact of Prompt Engineering on Efficiency, Code Quality, and Security in CRUD Application Development." Their expertise greatly influenced this study.
I also thank the academic and research communities whose work provided the foundation for this review.

Bibliography

Acher, M., Duarte, J. G., & Jézéquel, J.-M. (2023). On Programming Variability with Large Language Model-based Assistant. *Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume A*, 8–14. https://doi.org/10.1145/3579027.3608972

Ahmad, A., Waseem, M., Liang, P., Fahmideh, M., Aktar, M. S., & Mikkonen, T. (2023). Towards Human-Bot Collaborative Software Architecting with ChatGPT. *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, 279–285. https://doi.org/10.1145/3593434.3593468

Anuar, A. W., Kama, N., Azmi, A., & Rusli, H. M. (2023). A multivocal literature review on record management potential components in CRUD operation for web application development. *International Journal of Modeling, Simulation, and Scientific Computing*, *14*(02), 2341019. https://doi.org/10.1142/S1793962323410192

Bartalos, P., & Bieliková, M. (n.d.). *S)CRUD pattern support for semantic web applications.*

Biswas, S. (2023). Role of ChatGPT in Computer Programming. *Mesopotamian Journal of Computer Science,* pp. 8–16. https://doi.org/10.58496/MJCSC/2023/002

Bonteanu, A. M., & Tudose, C. (2024). Performance Analysis and Improvement for CRUD Operations in Relational Databases from Java Programs Using JPA, Hibernate, Spring Data JPA. *Applied Sciences*, *14*(7), 2743. https://doi.org/10.3390/app14072743

Botacin, M. (2023). GPThreats-3: Is Automatic Malware Generation a Threat? *2023 IEEE Security and Privacy Workshops (SPW)*, pp. 238–254. https://doi.org/10.1109/SPW59333.2023.00027

Chauvet, L. A., Cruz, D. M., & Quiroz, C. O. A. (n.d.). *ChatGPT as a Support Tool for Online Behavioral Task Programming*.

Chen, B., Zhang, Z., Langrene, N., & Zhu, S. (n.d.). *Unleashing the potential of prompt engineering in Large Language Models: A comprehensive review*.

Coello, C. E. A., Alimam, M. N., & Kouatly, R. (2024). Effectiveness of ChatGPT in Coding: A Comparative Analysis of Popular Large Language Models. *Digital*, *4*(1), 114–125. https://doi.org/10.3390/digital4010005

Diehl, P., Nader, N., Brandt, S., & Kaiser, H. (2024). *Evaluating AI-generated code for C++, Fortran, Go, Java, Julia, Matlab, Python, R, and Rust* (arXiv:2405.13101). arXiv. http://arxiv.org/abs/2405.13101

Dipongkor, A. K. (2024). Towards Interpreting the Behavior of Large Language Models on Software Engineering Tasks. *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, 255–257. https://doi.org/10.1145/3639478.3639798

Exploring the Competency of ChatGPT in Solving Competitive Programming Challenges. (2023). *International Journal of Advanced Trends in Computer Science and Engineering*, *13*(1), 13–23. https://doi.org/10.30534/ijatcse/2024/031312024

Feng, L., Yen, R., You, Y., Fan, M., Zhao, J., & Lu, Z. (2024). CoPrompt: Supporting Prompt Sharing and Referring in Collaborative Natural Language Programming. *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 1–21. https://doi.org/10.1145/3613904.3642212

Georgsen, R. E. (n.d.). *Beyond Code Assistance with GPT-4: Leveraging GitHub Copilot and ChatGPT for Peer Review in VSE Engineering*.

Gerosa, M., Trinkenreich, B., Steinmacher, I., & Sarma, A. (2024). Can AI serve as a substitute for human subjects in software engineering research? *Automated Software Engineering*, *31*(1), 13. https://doi.org/10.1007/s10515-023-00409-6

Gómez, O. S., Rosero, R. H., & Cortés-Verdín, K. (2020). CRUDyLeaf: A DSL for Generating Spring Boot REST APIs from Entity CRUD Operations. *Cybernetics and Information Technologies*, *20*(3), 3–14. https://doi.org/10.2478/cait-2020-0024

Hamer, S., d'Amorim, M., & Williams, L. (2024). *Just another copy and paste? Comparing the security vulnerabilities of ChatGPT generated code and StackOverflow answers* (arXiv:2403.15600). arXiv. http://arxiv.org/abs/2403.15600

Haque, Md. A., & Li, S. (2023). The Potential Use of ChatGPT for Debugging and Bug Fixing. *EAI Endorsed Transactions on AI and Robotics*, *2*(1), e4. https://doi.org/10.4108/airo.v2i1.3276

Hemil Patel & Shivam Parmar. (2024). *Prompt Engineering For Large Language Model*. https://doi.org/10.13140/RG.2.2.11549.93923

Huang, Y., Chen, Y., Chen, X., Chen, J., Peng, R., Tang, Z., Huang, J., Xu, F., & Zheng, Z. (2024). *Generative Software Engineering* (arXiv:2403.02583). arXiv. http://arxiv.org/abs/2403.02583

Husain, A. (2024). Potentials of ChatGPT in Computer Programming: Insights from Programming Instructors. *Journal of Information Technology Education: Research*, pp. *23*, 002. https://doi.org/10.28945/5240

Jaber, M. A., Beganović, A., & Almisreb, A. A. (2023). *Methods and Applications of ChatGPT in Software Development: A Literature Review*. *12*(1).

Jiang, E., Olson, K., Toh, E., Molina, A., Donsbach, A., Terry, M., & Cai, C. J. (2022). PromptMaker: Prompt-based Prototyping with Large Language Models. *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 1–8. https://doi.org/10.1145/3491101.3503564

Jin, K., Wang, C.-Y., Pham, H. V., & Hemmati, H. (2024). *Can ChatGPT Support Developers? An Empirical Evaluation of Large Language Models for Code Generation*. https://doi.org/10.1145/3643991.3645074

Khoury, R., Avila, A. R., Brunelle, J., & Camara, B. M. (2023). *How Secure is Code Generated by ChatGPT?* (arXiv:2304.09655). arXiv. http://arxiv.org/abs/2304.09655

Koubaa, A., Qureshi, B., Ammar, A., Khan, Z., Boulila, W., & Ghouti, L. (2023). Humans are still better than ChatGPT: Case of the IEEEXtreme competition. *Heliyon*, *9*(11), e21624. https://doi.org/10.1016/j.heliyon.2023.e21624

Kuhail, M. A., Mathew, S. S., Khalil, A., Berengueres, J., & Hussain, S. J. (2024). *"Will I Be Replaced?" Assessing ChatGPT's Effect on Software Development and Programmer Perceptions of AI Tools*.

Liu, J., Tang, X., Li, L., Chen, P., & Liu, Y. (2023). *Which is a better programming assistant? A comparative study between ChatGPT and stack overflow* (arXiv:2308.13851). arXiv. http://arxiv.org/abs/2308.13851

Liu, J., Xia, C. S., Wang, Y., & Zhang, L. (n.d.). *Is Your Code Generated by ChatGPT Really Correct?*

Liu, Y., Le-Cong, T., Widyasari, R., Tantithamthavorn, C., Li, L., Le, X.-B. D., & Lo, D. (2024). Refining ChatGPT-Generated Code: Characterizing and Mitigating Code Quality Issues. *ACM Transactions on Software Engineering and Methodology*, *33*(5), 1–26. https://doi.org/10.1145/3643674

Nascimento, N., Alencar, P., & Cowan, D. (2023). *Artificial Intelligence vs. Software Engineers: An Empirical Study on Performance and Efficiency using ChatGPT*.

Nashid, N., Sintaha, M., & Mesbah, A. (2023). Retrieval-Based Prompt Selection for Code-Related Few-Shot Learning. *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2450–2462. https://doi.org/10.1109/ICSE48619.2023.00205

Noy, S., & Zhang, W. (2023). *Experimental Evidence on the Productivity Effects of Generative Artificial Intelligence*.

Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E., Chou, R., Glanville, J., Grimshaw, J. M., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E. W., Mayo-Wilson, E., McDonald, S., … Moher, D. (2021). The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ*, n71. https://doi.org/10.1136/bmj.n71

Pantelelis, M., & Kalloniatis, C. (2022). Mapping CRUD to Events—Towards an object to event-sourcing framework. *Proceedings of the 26th Pan-Hellenic Conference on Informatics*, 285–289. https://doi.org/10.1145/3575879.3576006

PRISMA statement. (n.d.). PRISMA Statement. Retrieved 29 July 2024, from https://www.prisma-statement.org

Ricárdez, J. J. U., Vasconcelos, M. P., Domínguez, R. G., & Romero, F. C. (2024). *Integration of Generative AI with ChatGPT in Software Development*.

Rush, A. (2023). MiniChain: A Small Library for Coding with Large Language Models. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 311–317. https://doi.org/10.18653/v1/2023.emnlp-demo.27

Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2024). *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications* (arXiv:2402.07927). arXiv. http://arxiv.org/abs/2402.07927

Sakib, F. A., Khan, S. H., & Karim, A. H. M. R. (2023). *Extending the Frontier of ChatGPT: Code Generation and Debugging* (arXiv:2307.08260). arXiv. http://arxiv.org/abs/2307.08260

Sarkar, A. (2023). Will Code Remain a Relevant User Interface for End-User Programming with Generative AI Models? *Proceedings of the 2023 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pp. 153–167. https://doi.org/10.1145/3622758.3622882

Schröder, M., Hees, J., Bernardi, A., Ewert, D., Klotz, P., & Stadtmüller, S. (2018). *Simplified SPARQL REST API - CRUD on JSON Object Graphs via URI Paths* (Vol. 11155, pp. 40–45). https://doi.org/10.1007/978-3-319-98192-5_8

Shirafuji, A., Oda, Y., Suzuki, J., Morishita, M., & Watanobe, Y. (2023). Refactoring Programs Using Large Language Models with Few-Shot Examples. *2023 30th Asia-Pacific Software Engineering Conference (APSEC)*, 151–160. https://doi.org/10.1109/APSEC60848.2023.00025

Sobania, D., Briesch, M., Hanna, C., & Petke, J. (2023). *An Analysis of the Automatic Bug Fixing Performance of ChatGPT* (arXiv:2301.08653). arXiv. http://arxiv.org/abs/2301.08653

Sun, Y., Jang, E., Ma, F., & Wang, T. (2024). Generative AI in the Wild: Prospects, Challenges, and Strategies. *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 1–16. https://doi.org/10.1145/3613904.3642160

Taeb, M., Chi, H., & Bernadin, S. (2024). Assessing the Effectiveness and Security Implications of AI Code Generators. *Journal of The Colloquium for Information Systems Security Education, 11*(1), 6. https://doi.org/10.53735/cisse.v11i1.180

Truica, C.-O., Radulescu, F., Boicea, A., & Bucur, I. (2015). Performance Evaluation for CRUD Operations in Asynchronously Replicated Document Oriented Database. *2015 20th International Conference on Control Systems and Computer Science*, 191–196. https://doi.org/10.1109/CSCS.2015.32

Vaillant, T. S., de Almeida, F. D., Neto, P. A. M. S., Gao, C., Bosch, J., & de Almeida, E. S. (2024). *Developers' Perceptions on the Impact of ChatGPT in Software Development: A Survey* (arXiv:2405.12195). arXiv. http://arxiv.org/abs/2405.12195

Weidinger, L., Mellor, J., Rauh, M., Griffin, C., Uesato, J., Huang, P.-S., Cheng, M., Glaese, M., Balle, B., Kasirzadeh, A., Kenton, Z., Brown, S., Hawkins, W., Stepleton, T., Biles, C., Birhane, A., Haas, J., Rimell, L., Hendricks, L. A., … Gabriel, I. (2021). *Ethical and social risks of harm from Language Models* (arXiv:2112.04359). arXiv. http://arxiv.org/abs/2112.04359

Weisz, J. D., Muller, M., He, J., & Houde, S. (2023). *Toward General Design Principles for Generative AI Applications* (arXiv:2301.05578). arXiv. http://arxiv.org/abs/2301.05578

Wermelinger, M. (2023). Using GitHub Copilot to Solve Simple Programming Problems. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 172–178. https://doi.org/10.1145/3545945.3569830

Yetiştiren, B., Özsoy, I., Ayerdem, M., & Tüzün, E. (2023). *Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT* (arXiv:2304.10778). arXiv. http://arxiv.org/abs/2304.10778

Yin, X. (2024). *Pros and Cons! Evaluating ChatGPT on Software Vulnerability* (arXiv:2404.03994). arXiv. http://arxiv.org/abs/2404.03994

Yu, X., Liu, L., Hu, X., Keung, J. W., Liu, J., & Xia, X. (2024). *Fight Fire with Fire: How Much Can We Trust ChatGPT on Source Code-Related Tasks?* (arXiv:2405.12641). arXiv. http://arxiv.org/abs/2405.12641

Zhang, C., Liu, H., Zeng, J., Yang, K., Li, Y., & Li, H. (2024). *Prompt-Enhanced Software Vulnerability Detection Using ChatGPT* (arXiv:2308.12697). arXiv. http://arxiv.org/abs/2308.12697