Exploring Early Adopters' Perceptions of ChatGPT as a Code Generation Tool

Gian Luca Scoccia
Gran Sasso Science Institute
L'Aquila, Italy
gianluca.scoccia@gssi.it

***Abstract*—ChatGPT is an artificial intelligence chatbot developed by OpenAI, able of interacting in a conversational way by taking into account successive input prompts. Among many possible uses, ChatGPT has been found to possess code generation capabilities, being able to generate code snippets and assist developers in their programming tasks. This paper performs a qualitative exploration of perceptions of early adopters regarding the use of ChatGPT for code generation, acknowledging the substantial impact this tool can have in the software development landscape.**

We collected a diverse set of discussions from early adopters of ChatGPT code generation capabilities and, leveraging an open card sorting methodology categorized it into relevant topics with the goal of extracting insights into the experiences, opinions, and challenges they faced. We found that early adopters *(i)* report their own mixed usage experiences, *(ii)* share suggestions for prompt engineering, *(iii)* debate the extent to which they can trust generated code, and *(iv)* discuss the impact that ChatGPT can have on the software development process. We discuss the implications of the insights we extracted from early adopters' perspectives and provide recommendations for future research.

***Index Terms*—Artificial intelligence, Code generation, ChatGPT.**

I. Introduction

ChatGPT, developed by OpenAI, is an artificial intelligence chatbot designed to engage in natural language conversations. Relying on the Generative Pre-trained Transformer (GPT) architecture [1], it is able to comprehend input prompts in natural language and generate human-like text responses. As a general-purpose model, ChatGPT can be employed for a variety of tasks, including creative writing, translation, summarization, and question-answering [2]. Notably, ChatGPT can be used for *code generation*, i.e., the process of automatically producing a complete program from high-level instructions or specifications [3], [4]. An example of these code-generation capabilities is provided in Figure 1, in which ChatGPT generated a complete program in response to a simple input prompt (i.e., *"Write a program to test if a number is prime in JavaScript"*).

Due to its ease of use and its promising code generation performance, ChatGPT has attracted considerable attention from software professionals. In this paper, we perform a qualitative exploration of perceptions of early adopters regarding the use of ChatGPT for code generation, acknowledging the substantial impact this tool can have in the software development landscape. Specifically, we collected a diverse set of discussions from early adopters of ChatGPT for code generation and, leveraging an open card sorting methodology, categorized it into relevant topics with the goal of extracting insights into the experiences, opinions, and challenges they faced. Our results show that early adopters discuss four main topics related to the usage of ChatGPT for code generation: (i) their own (both positive and negative) usage experiences, (ii) suggestions for improving the quality of generated results via structured prompts (i.e., *prompt engineering*), (iii) the extent to which they can trust the correctness of generated code, and (iv) the impact that ChatGPT can have on the software development landscape. Building on the insights we extracted from early adopters' perspectives, we highlight research directions for which further work is needed to ensure that future code generators are usable, safe, and inclusive.

The remainder of this paper is structured as follows. Section II introduces preliminary concepts. Section III describes the study design. Section IV illustrates the obtained results, and they are discussed in Section V. Section VI describes the threats to the validity of the study, while Section VII discusses related work. Section VIII closes the paper.

II. Background

ChatGPT is an artificial intelligence chatbot developed by OpenAI. Released on the 30$^{\text{th}}$ of November 2022, it was originally based on version 3.5 of the GPT (Generative Pre-trained Transformer) large language model [1], while newer versions rely on version 4 of the same model. Differently from other large language models (LLMs), ChatGPT has the capability of interacting in a conversational way, by taking into account successive input prompts and generated replies.

These capabilities stem from ChatGPT training process, during which it has been fine-tuned for conversational applications using a combination of supervised learning [5] and reinforcement learning from human feedback (RLHF) [6]. ChatGPT is versatile and, among its many possible uses, it can be leveraged for creative writing, translation, summarization, question-answering, and (notably) to write and debug computer programs [2], [7]. These capabilities are further augmented by a plugin system that adds advanced features like code interpretation [8]. However, usage of ChatGPT is not without risks as, akin to other LLMs, it can produce wrong answers and *hallucinate*, i.e., generate plausible but factually wrong answers that are particularly hard to identify for human

readers [9]. The extent to which ChatGPT can generate correct code is still under investigation, although, in a recent study [7], ChatGPT is on par with other state-of-the-art approaches for program repair.

III. Study design

Our study is exploratory in its nature, with the overall *goal* of extracting insights into the experiences, opinions, and challenges faced by early adopters of ChatGPT code-generation capabilities. This, in turn, will help in *(i)* assessing the impact of ChatGPT on code development activities, *(ii)* provide evidence on the issues experienced by early adopters, and *(iii)* highlight possible future research directions. We refine our goal into the following research question:

RQ - What are the main topics discussed by early adopters of ChatGPT for code generation?

To answer this research question, as previously done in the software engineering literature [10]–[12], we extract knowledge from online communities. Specifically, we collect and analyze discussions on the use of ChatGPT for code-generation hosted on the popular website Hacker News[1]. We chose Hacker News for multiple reasons. First, *(i)* it has an active technical audience, and timely discussions of innovative tools and methodologies (hence, users of Hacker News can be considered early-adopters); Second, *(ii)* it enforces strict moderation guidelines and adopts a comment ranking system, both of which help to enhance the overall quality of comments and ensure the absence of spam. Third, *(iii)* it's been previously used in other software engineering studies [13]–[17], providing further confidence in the quality of its discussions. Finally, *(iv)* it has a public API, which simplifies the data collection process.

A. Data collection

We collected relevant discussion data from Hacker News using its search API[2]. For this purpose, we queried the API using the keywords *"chat"*, *"gpt"*, and *"code"* on the 15th of June 2023. These were selected after experimentation with different combinations of keywords and all three have to be included in a comment in order for it to be returned by the query. This resulted in the identification of 3,914 distinct comments. However, the API only allows for the download of 1,000 results, so we considered this number for further analysis. While this is a limiting factor, we felt that this would be a sufficient number for a preliminary study. Moreover, from an initial exploration of the data, we were confident that our choice of keywords was effective in identifying relevant comments while also limiting the number of false positives present in the results. To improve the accuracy of our analysis, we filtered out from collected comments those with a length of less than twenty words, as these are likely to be uninformative for our purposes. We then randomly sampled a subset of 100 comments for our analysis procedure described below.

[1]https://news.ycombinator.com/
[2]https://hn.algolia.com/

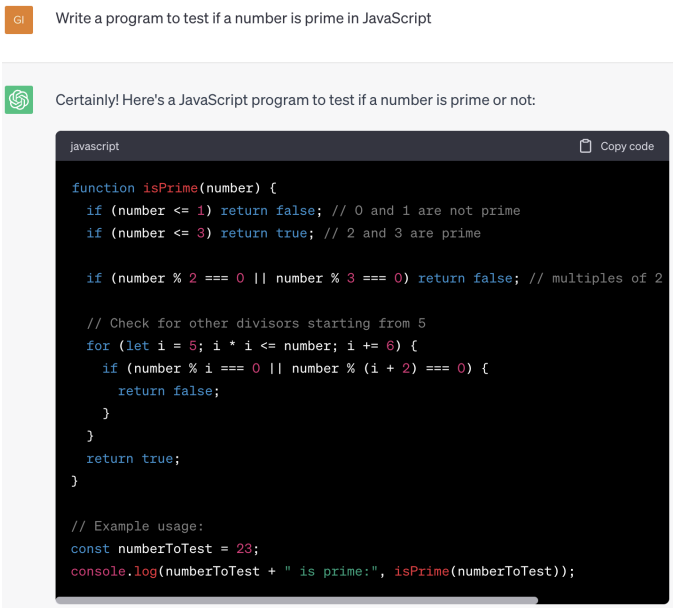

Fig. 1. An example program generated by ChatGPT from a textual prompt

B. Data analysis

To analyze collected data, the author performed a *card sorting* process [18], [19] to identify categories present in the data. Card sorting is a commonly used technique to derive categories from data [20]. There are three types of card sorting, namely *closed card sorting* with predefined categories for data, *open card sorting* with no predefined categories, and *hybrid card* sorting which combines the previous two types [21]. Given the exploratory nature of our study, we adopted an open card sorting process. During our analysis, 14 comments were deemed unrelated to the use of ChatGPT for code generations and thus discarded from the analysis, as false positives.

IV. Results

The analysis described in the previous section resulted in the identification from the data of 4 main discussed topics: *User experiences*, *Prompt engineering*, *Trust*, and *Impact on the software development process*. We describe each theme in the following, making use of illustrative examples. Each example is complete with the comment unique `objectID`, to allow for identification in the dataset.

User experiences - This theme encompasses comments in which users report their experiences while using ChatGPT for code generation purposes. Notably, we identified varied ways of employing ChatGPT code-generation capabilities. For instance, one comment reports a positive experience while using it as guidance during coding activities:

> *" The code it generates for me is usually not perfect, but for me it provides conceptual insights that have gotten me unstuck from gnarly situations. [...] I feel like it's more like a senior engineer scribbling on a whiteboard—no, it's not perfect code, but it's the right idea." (36087083)*

Similarly, another user reports a positive experience while using it to generate code in unfamiliar programming languages:

> *"I find chat GPT to be very helpful for working with programming languages that I'm less comfortable using (shell, python). I know enough to evaluate correct code in these languages, but producing it from scratch is more difficult, which seems like a sweet spot for carefully using ChatGPT for code. [...]" (36091502)*

A different user reports using ChatGPT to quickly obtain jump-start their own projects:

> *"[...] I wouldn't be able to implement something from scratch without googling or copying stuff from Github. When I ask ChatGPT to do it for me it gives me an excellent starting point. Sure, there will be bugs, but because I know what I want I can spot and fix them immediately. Its much faster to adjust ChatGPT's code than it is to Google around for starting points." (35848830)*

On the other end of the spectrum, other users report negative experiences while using ChatGPT for code generation:

> *"Very cool. But nearly every time I've asked ChatGPT for code, it gives me either plausible-but-wrong code, or total hallucinations (flags and APIs that could exist, but don't)." (35765407)*

Oftentimes, large language model errors are a consequence of *hallucinations*, i.e., factually incorrect content presented as a fact [9]. In the context of code generation, errors of this kind can be particularly hard to detect, resulting in a frustrating experience:

> *"I burned an hour with chat gpt insisting an AddOrUpdate function existed in microsoft entity framework. When I called bullshit it hallucinated that another library contained it. Then it hallucinated versions... Then I gave up... This has been my experience with ChatGPT and code,it hallucinates a lot of stuff. [...]" (34703814)*

Prompt engineering - This theme groups together comments that provide suggestions on how to write prompts (i.e., text provided as input) to optimize the results generated by ChatGPT. Prompt engineering is a novel concept and it is still under active investigation by researchers and practitioners [22]–[24]. During our analysis, we identified comments in which users share suggestions for prompt engineering based on their own experiences:

> *"Using ChatGPT to code effectively requires internalizing the small sliding context window and correct specification of the problem-space. You can get real work done if you thoughtfully split your project into parts, and you can save time by jumping back to a contextual fork where your program was fully specified." (35853949)*

Contextually, we identified opinions that highlight difficulties related to having to specify the code to be generated using natural language:

> *"Those beautiful English paragraphs "telling" ChatGPT what code to create are worse than programming. Not all of us programmers are native English speakers, and, as such, we might miss some of the nuances of the English language itself and thus fail to get the most out of ChatGPT. [...]" (35279267)*

Trust - This theme collects comments that discuss the implications of confiding in (potentially faulty) generated code. Some users are uncomfortable in including ChatGPT generated code in real-world products without a prior review or verification process:

> *"What are one of the professional use cases where you would just feel comfortable YOLOing some ChatGPT generated code into production? Publishing a journal without verification? You should also take note of the warnings in the GPT-4 manual, it's a much more convincing liar than GPT-3. Quite explicitly says that. [...]" (34830565)*

However, we also identified the opinions of users that are unconcerned about the way the code is produced:

> *"I would trust ChatGPT code about as much as I trust the code produced by any human. All the Therac-25[3] code was written by a human, so what is the argument here exactly? [...] I like to think that it is not about who (or what) writes the code in the first place, it is about the review and testing procedures that ensure the quality of the final product." (35387398)*

Impact on the software development process - Comments grouped in this theme report impressions on what will be the impact caused by ChatGPT on the software development process and on related professions (e.g., software developer, software engineer). We identified comments that express disbelief in a major impact on the software industry as they believe ChatGPT cannot replace software professionals in a multitude of tasks:

> *"What the articles like this miss is that we understand pretty well how ChatGPT writes code. But what about maintenance, and particularly debugging? How would it make these tasks. And we know that these things take much much more time than writing code." (35356434)*

On the other hand, others believe that ChatGPT (and other large language models) will soon be able to replace software developers, especially in domains that are not safety-critical:

[3]The author of this comment is referring to the well-known case study of the Therac-25 safety-critical system [25].

"I think it could go either way depending on the product. For example in app/game/web development where code quantity > quality, hire more juniors who can bust out code with ChatGPT all day. But if you're developing software for medical devices, vehicle control systems, HFT, etc. Then nobody's going to let some college grad using ChatGPT touch it. You'd hire senior engineers who can be responsible for the reliable operation of the software and they can use ChatGPT for code review, test suites, etc." (35205999)

V. Discussion

We discuss in the following the implications of our results, presented in the previous section.

During our analysis, we identified **diversified usages** of ChatGPT code-generation capabilities. Indeed, unlike other tools that have well-defined use cases, ChatGPT is versatile and can be employed for varied purposes. This, however, means that it is difficult to provide a general measure of its performance, due to the high variability of contexts and tasks where it can be employed. Therefore, as observed, developers often have to experiment on their own for use cases of interest. This highlights the need for software engineering research of developing benchmarks and performing experiments for the evaluation of ChatGPT capabilities in varied software development-related activities.

We observed several instances of developers discussing **prompt engineering** practices. At their core, these practices attempt to introduce structure in otherwise unstructured input prompts, to improve the quality of generated outputs. In fact, it's known that techniques such as *chain-of-thought* prompting improve the ability of large language models to perform complex tasks [23]. Nonetheless, fully mastering these techniques is non-trivial, due to the non-determinism intrinsic in LLMs [26] and the fact given that even small variations in input prompts can lead to significant changes in the generated output. Indeed, we identified comments against the use of textual prompts for code generation from users that believe to not be sufficiently proficient in natural language and, hence, believe to be at a disadvantage.

An important consideration is that ChatGPT has the potential to **democratize code** generation, allowing even non-specialists to create working programs from prompts in natural language. This, in turn, reduces the barrier of entry for non-experts to build software-based products. While this is undoubtedly beneficial, it also introduces potential risks, as non-specialized developers (or *prompters*), might then lack the necessary skill set to assess non-trivial functional and non-functional properties of generated code. Hence, the risk is the one of a future in which code will be even more pervasive than it is today (and already it has *"eaten the world"* [27]), but, on average, of lower quality. This not only results in an overall worsened experience but could expose end-users to privacy, security, and safety-related risks. Indeed, during our analysis, we identified several comments that show that developers are aware of potential quality issues in generated code. However, it is uncertain how these concerns hold up when other constraints are present (e.g., time-to-market). Hence, in the context of software engineering research, there is a need for novel techniques for the verification of functional and non-functional properties in generated code, that can efficiently scale and be usable by non-specialized developers.

VI. Threats to validity

In this section, we list the threats to the validity of our work and the actions taken to mitigate them.

Internal validity refers to the causality relationship between treatment and outcome [28]. In our study, we employed a manual card-sorting process to identify topics discussed by early adopters. As with all kinds of manual processes, there is a risk of potential mistakes introduced during the procedure. To mitigate this risk, we only reported the main topics identified, each substantiated by multiple comments.

External validity deals with the generalizability of obtained results [28]. Our study is preliminary in its nature and concerns a topic subject to rapid evolution. Hence, some of the identified impressions and criticisms expressed by users might no longer apply to newer versions of ChatGPT and other newly released large-language models. In addition, we manually analyzed a subset of 100 comments out of the 3,914 identified. As such, there is a risk that the sampled comments are not representative of the general population. To mitigate these threats we only reported the main themes we identified, each found in multiple comments. Moreover, our choice of Hacker News as a data source might pose a threat to the generalizability of obtained results since, as previously discussed, we believe that users of this discussion forum have above-average technical aptness. Hence, for future work, we plan on replicating our study on other discussion websites (e.g., Reddit) to validate and extend our results.

Construct validity concerns the relation between theory and observation [28]. We mitigated potential construct validity threats by defining all details related to the design of our study (e.g., the goal, research questions, hypotheses, and analysis procedures) before starting the study execution.

Conclusion validity deals with issues that affect the ability to draw the correct conclusions from the outcome of experiments [28]. To mitigate this threat, we report several illustrative examples for each identified topic, to substantiate our interpretation of the results and highlight identified concerns.

VII. Related work

In the following, we briefly survey other studies related to our own.

At the time of writing, due to its novelty, limited scientific literature is available on the topic of ChatGPT for code generation. Feng and colleagues [29] conducted a quantitative study of usages of ChatGPT, collecting samples from social media websites Reddit and Twitter. Analyzing collected data,

they found that ChatGPT is commonly used for a diverse range of tasks in more than 10 programming languages, with Python and JavaScript being the two most popular. Moreover, by performing sentiment analysis of collected social media posts, they report that fear is the dominant emotion associated with ChatGPT's code generation, overshadowing other emotions. Our study complements theirs, with a qualitative investigation of experiences, opinions, and challenges faced by early adopters. Sobania [7] et al. investigated the bug-fixing performance of ChatGPT, comparing it with other approaches reported in the literature. They highlight that ChatGPT's bug-fixing performance is comparable to other deep learning approaches used in the literature. Moreover, they show that interacting with follow-up prompts further increases the success rate of ChatGPT, outperforming state-of-the-art. In our study, we report evidence that developers are aware that prompt engineering can greatly impact the quality of produced code.

Researchers have investigated how other LLM-based code generation tools are used by programmers and their impact on code development activities [4], [17], [30], [31]. Vaithilingam and collegues [4] conducted a user study with 24 participants to understand how programmers use and perceive GitHub Copilot, an LLM-based code generation tool. They report that Copilot did not necessarily improve the task completion time or success rate, but participants welcomed the use of the tool, as it often provided a useful starting point. In a related study, Barke et al. [31] conducted a grounded theory study of how programmers interact with Copilot, observing 20 participants as they solved diverse programming tasks. They hypothesize that interactions with programming assistants are bimodal: in *acceleration mode*, the programmer knows what to do and leverages the assistant to get there faster, while in *exploration mode*, the programmer is unsure how to proceed and the assistant is used to explore their options. Sarkar et al. [17] compiled observations from the above user studies and additionally gathered experience reports of programming assistants usage from Hacker News. The compiled observations complement our own, reporting that prompting is hard, validation is important, and programmers often use code assistants for boilerplate, reusable code. Liu and colleagues [30] designed a system in which code generated by an LLM is translated back into structured natural language. In a follow-up user study, they assessed that this reverse translation improves end-users' understanding of the scope and capabilities of the code-generating model, and the kind of language needed to use it effectively. In our study, we collected user feedback *in-the-wild*, rather than in a controlled experiment, and investigated the topics discussed, including user experiences with the model and prompt engineering practices.

VIII. Conclusion and future work

We conducted a qualitative investigation of the perceptions of early adopters of ChatGPT for code generation. For this purpose, we identified 3,914 comments on the social news website Hacker News that discuss the use of ChatGPT for code generation and analyzed a sample of 100 comments employing a card sorting process. Our analysis resulted in the identification of four main topics in the discussion data: *User experiences*, *Prompt engineering*, *Trust*, and *Impact on the software development process*. We reported several examples of comments in each of the four categories and, building from extracted insights, identified directions for future research toward the development of usable, safe, and inclusive future code generators.

In future work, we will investigate how to automate the performed analysis (e.g., through machine learning techniques) to extend it to the complete dataset. Moreover, we plan to expand the analysis to other discussion websites (e.g., Reddit). Finally, we will focus more closely on selected aspects (e.g., usage of ChatGPT for exploration rather than acceleration [31]) to deepen the extracted insights.

References

[1] OpenAI, "Introducing ChatGPT," https://openai.com/blog/chatgpt, Nov 2022, accessed: 2023-07-20.

[2] K. Roose, "The brilliance and weirdness of ChatGPT," *The New York Times*, 2022.

[3] B. Wei, G. Li, X. Xia, Z. Fu, and Z. Jin, "Code generation as a dual task of code summarization," *Advances in neural information processing systems*, vol. 32, 2019.

[4] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *Chi conference on human factors in computing systems extended abstracts*, 2022, pp. 1–7.

[5] S. J. Russell, *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[6] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.

[7] D. Sobania, M. Briesch, C. Hanna, and J. Petke, "An analysis of the automatic bug fixing performance of ChatGPT," *arXiv preprint arXiv:2301.08653*, 2023.

[8] OpenAI, "ChatGPT plugins," https://openai.com/blog/chatgpt-plugins, March 2023, accessed: 2023-07-20.

[9] N. Dziri, S. Milton, M. Yu, O. Zaiane, and S. Reddy, "On the origin of hallucinations in conversational models: Is it the datasets or the models?" in *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2022.

[10] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "A systematic mapping study of software development with GitHub," *Ieee access*, vol. 5, pp. 7173–7192, 2017.

[11] S. Meldrum, S. A. Licorish, and B. T. R. Savarimuthu, "Crowdsourced knowledge on Stack Overflow: A systematic mapping study," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, 2017, pp. 180–185.

[12] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming q&a in stackoverflow," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE, 2012, pp. 25–34.

[13] T. Barik, B. Johnson, and E. Murphy-Hill, "I heart Hacker News: expanding qualitative research findings by analyzing social news websites," in *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, 2015, pp. 882–885.

[14] G. Schermann, J. Cito, P. Leitner, U. Zdun, and H. C. Gall, "We're doing it live: A multi-method empirical study on continuous experimentation," *Information and Software Technology*, vol. 99, pp. 41–57, 2018.

[15] G. Brito, T. Mombach, and M. T. Valente, "Migrating to GraphQL: A practical assessment," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 140–150.

[16] P. Leitner, E. Wittern, J. Spillner, and W. Hummer, "A mixed-method empirical study of function-as-a-service software development in industrial practice," *Journal of Systems and Software*, vol. 149, pp. 340–359, 2019.

[17] A. Sarkar, A. D. Gordon, C. Negreanu, C. Poelitz, S. S. Ragavan, and B. Zorn, "What is it like to program with artificial intelligence?" *arXiv preprint arXiv:2208.06213*, 2022.

[18] J. R. Wood and L. E. Wood, "Card sorting: current practices and beyond," *Journal of Usability Studies*, vol. 4, no. 1, pp. 1–6, 2008.

[19] D. Spencer, *Card sorting: Designing usable categories*. Rosenfeld Media, 2009.

[20] M. Ntouvaleti and C. Katsanos, "Validity of the open card sorting method for producing website information structures," in *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 2022, pp. 1–7.

[21] T. Zimmermann, "Card-sorting: From text to themes," in *Perspectives on data science for software engineering*. Elsevier, 2016, pp. 137–141.

[22] L. Reynolds and K. McDonell, "Prompt programming for large language models: Beyond the few-shot paradigm," in *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, 2021, pp. 1–7.

[23] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[24] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. V. Le *et al.*, "Least-to-most prompting enables complex reasoning in large language models," in *The Eleventh International Conference on Learning Representations*, 2022.

[25] N. G. Leveson and C. S. Turner, "An investigation of the Therac-25 accidents," *Computer*, vol. 26, no. 7, pp. 18–41, 1993.

[26] C. W. Johnson, "The increasing risks of risk assessment: On the rise of artificial intelligence and non-determinism in safety-critical systems," in *the 26th Safety-Critical Systems Symposium*. Safety-Critical Systems Club York, UK., 2018, p. 15.

[27] M. Andreessen, "Why software is eating the world," *Wall Street Journal*, vol. 20, no. 2011, p. C2, 2011.

[28] T. D. Cook, D. T. Campbell, and A. Day, *Quasi-experimentation: Design & analysis issues for field settings*. Houghton Mifflin Boston, 1979, vol. 351.

[29] Y. Feng, S. Vanam, M. Cherukupally, W. Zheng, M. Qiu, and H. Chen, "Investigating code generation performance of chat-gpt with crowdsourcing social data," in *Proceedings of the 47th IEEE Computer Software and Applications Conference*, 2023, pp. 1–10.

[30] M. X. Liu, A. Sarkar, C. Negreanu, B. Zorn, J. Williams, N. Toronto, and A. D. Gordon, ""What it wants me to say": Bridging the abstraction gap between end-user programmers and code-generating large language models," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023, pp. 1–31.

[31] S. Barke, M. B. James, and N. Polikarpova, "Grounded copilot: How programmers interact with code-generating models," *Proceedings of the ACM on Programming Languages*, vol. 7, no. OOPSLA1, pp. 85–111, 2023.