Journal of Information Technology Education: Innovations in Practice

An Official Publication of the Informing Science Institute InformingScience.org

JITEiip.org

Volume 23, 2024

Coding with AI as an Assistant: Can AI Generate Concise Computer Code?

Andrew Millam	Grand Canyon University, Phoenix, AZ, USA	AMillam@my.gcu.edu
Christine Bakke*	Grand Canyon University, Phoenix, AZ, USA	Christine.Bakke@gcu.edu

* Corresponding author

Abstract

Aim/Purpose	This paper is part of a multi-case study that aims to test whether generative AI makes an effective coding assistant. Particularly, this work evaluates the ability of two AI chatbots (ChatGPT and Bing Chat) to generate concise computer code, considers ethical issues related to generative AI, and offers suggestions for how to improve the technology.
Background	Since the release of ChatGPT in 2022, generative artificial intelligence has steadily gained wide use in software development. However, there is conflicting information on the extent to which AI helps developers be more productive in the long term. Also, whether using generated code violates copyright restrictions is a matter of debate.
Methodology	ChatGPT and Bing Chat were asked the same question, their responses were recorded, and the percentage of each chatbot's code that was extraneous was calculated. Also examined were qualitative factors, such as how often the generated code required modifications before it would run.
Contribution	This paper adds to the limited body of research on how effective generative AI is at aiding software developers and how to practically address its shortcomings.
Findings	Results of AI testing observed that 0.7% of lines and 1.4% of characters in ChatGPT's responses were extraneous, while 0.7% of lines and 1.1% of characters in Bing Chat's responses were extraneous. This was well below the 2% threshold, meaning both chatbots can generate concise code. However, code from both chatbots frequently had to be modified before it would work; ChatGPT's code needed major modifications 30% of the time and minor ones

Accepting Editor Keith A. Willoughby │ Received: March 16, 2024 │ Revised: June 12, July 29, 2024 │ Accepted: August 10, 2024.
Cite as: Millam, A., & Bakke, C. (2024). Coding with AI as an assistant: Can AI generate concise computer code?. *Journal of Information Technology Education: Innovations in Practice, 23,* Article 9. https://doi.org/10.28945/5362

	50% of the time, while Bing Chat's code needed major modifications 10% of the time and minor ones 70% of the time.
Recommendations for Practitioners	Companies building generative AI solutions are encouraged to use this study's findings to improve their models, specifically by decreasing error rates, adding more training data for programming languages with less public documentation, and implementing a mechanism that checks code for syntactical errors. Developers can use the findings to increase their productivity, learning how to reap generative AI's full potential while being aware of its limitations.
Recommendations for Researchers	Researchers are encouraged to continue where this paper left off, exploring more programming languages and prompting styles than the scope of this study allowed.
Impact on Society	As artificial intelligence touches more areas of society than ever, it is crucial to make AI models as accurate and dependable as possible. If practitioners and researchers use the findings of this paper to improve coders' experience with generative AI, it will make millions of developers more productive, saving their companies money and time.
Future Research	The results of this study can be strengthened (or refuted) by a future study with a large, diverse dataset that more fully represents the programming languages and prompting styles developers tend to use. Moreover, further research can examine the reasons generative AI fails to deliver working code, which will yield valuable insights into improving these models.
Keywords	artificial intelligence, Bing Chat, ChatGPT, code modifications, concise code, ethics, programming assistants

INTRODUCTION

Since the invention of high-level programming languages, software has become increasingly powerful and sophisticated. Today, it is not uncommon for a single application to consume gigabytes of random access memory and utilize several processor cores at once. As programs continue to get more complex, new technologies assist developers so they can complete their jobs more efficiently and spend less time debugging code. One such technology is generative artificial intelligence, which is still in its beginning stages but already showing immense potential to empower developers so they can work more productively than ever.

Nevertheless, generative AI frequently falls short. Many developers have expressed frustration when it fails to produce a satisfactory result (Zhou et al., 2023). There are also concerns that code generated by GitHub Copilot and similar tools may contain security vulnerabilities (Perry et al., 2023). Additionally, there have been lawsuits against AI companies because their chatbots often eject copyrighted code line-for-line without giving credit to the programmers who wrote it (Caballar, 2022; Samuelson, 2023).

There are countless research papers on the topic of generative AI. Some are theoretical in nature, describing what AI is (or may soon be) capable of. Others look at how developers are using AI to do their jobs. Still, others focus on the shortcomings of present AI models. However, none was found to address all three research goals of this paper adequately: whether generative AI is useful to programmers, how the technology can be improved, and how ethical issues shape the AI landscape.

The paper highlights a single case as part of a larger multi-case project aiming to answer the research question: "Does AI make a good coding assistant?" In the project, students developed software for a remote client who wanted to automate the process of finding and replacing names of characters in children's books for custom names. They wrote this application with the help of AI assistants. Ten projects were designed in parallel, with punctuated feedback from the client during the initial design and testing phases. Students collected data throughout the project to determine the value of AI as a coding assistant, each person studying a unique metric that contributed to that end.

This paper examines the effectiveness of generative AI in producing concise computer code. Particularly, it compares ChatGPT's and Bing Chat's ability to generate concise code, along with the number of tries it takes to get a working program and whether the code requires modifications to run.

The paper will begin by defining important terms used in this study. Then, it will state the research question, hypotheses, and objectives of the study. Next, it will explore existing literature on generative AI in software production. After that, the methodology used to test the hypothesis will be laid out, followed by the data collected and the analysis of that data. Finally, it will investigate ethical considerations of AI code generation and conclude with the implications of this study's findings. At the very end are two appendices containing raw data from the collection and analysis stages.

DEFINITIONS

1. **Chatbots** refer to the two generative AI models assessed in this study, namely ChatGPT and Bing Chat.
2. **Concise code** means computer code that contains few to no extraneous lines or characters. Since programming styles vary, concise code does not necessarily entail the shortest code possible; rather, it implies that the code cannot be made shorter within the same style.
3. **Extraneous code** consists of unused imports and variables, trailing whitespace, and any lines or sections that can be made shorter without negatively affecting the code's readability or efficiency.
4. **Generative AI** encompasses all text-based large language models that can produce computer code in response to a natural language prompt. In some contexts, generative AI specifically means ChatGPT and Bing Chat, the two chatbots examined in this paper.
5. **Major modifications** imply that a code sample from a generative AI model would not compile or execute without significant effort to manually fix errors. For example, the code may depend on obsolete libraries, or the solution may not be fully implemented.
6. **Minor modifications** mean that a code sample from a generative AI model either (a) would not execute but took little effort to manually debug or (b) contained trivial errors and did not produce the expected result. For instance, the chatbot may have omitted an "import" statement, or parameters may be missing from a function call.

RESEARCH HYPOTHESES AND OBJECTIVES

This paper is one part of a multi-case study attempting to answer the question, "Does AI make a good coding assistant?" Readers interested in the findings of other student researchers involved in this multi-case study are encouraged to reach out to Dr Christine Bakke, coauthor of Bakke and Sakai (2022). Other students explored different research questions, including "How often does AI-generated code correctly compile and execute?" and "Can AI generate efficient computer code?" However, only some students chose to publish their results.

This paper addresses the ability of generative AI to write concise computer code. Specifically, it evaluates the following research question: "Can ChatGPT and Bing Chat generate concise computer code better than a human developer could?" This question has two parts. First, can either of the chatbots outperform a human programmer in producing concise code? Second, which chatbot will perform better in that context: ChatGPT or Bing Chat?

The authors propose two hypotheses, one for each part of the research question. The first hypothesis is that both chatbots will write less concise code than an experienced developer could because they are still relatively new and under active research. If one or both chatbots consistently write code equally concisely compared to what a human programmer can write, then this hypothesis will be rejected. The second hypothesis is that Bing Chat will write concise computer code better than ChatGPT will because Bing employs GPT-4 and can pull content from the Internet with retrieval augmented generation (RAG), while the free version of ChatGPT uses GPT-3.5 and cannot access up-to-date web content. If there is no significant difference in quality between Bing Chat and ChatGPT, this hypothesis will be disproven.

LITERATURE

Although there are presently no peer-reviewed journal articles that cover the same ground as this study, numerous articles discuss generative AI in software development in general. Here are six examples:

- Bankins et al. (2024) study artificial intelligence in the workplace. They examine five factors on how employees use AI and how it impacts their productivity: how humans collaborate with AI, how workers perceive the capabilities of AI versus what human developers can do, employees' attitudes toward AI, algorithmic management in platform-based work (that is, having AI monitor and manage gig workers), and how utilizing AI impacts the labor market. Altogether, the authors consider 68 articles on those five topics. Though quite detailed, the paper does not address whether AI makes a good coding assistant; instead, it discusses how workers interact with AI.
- Campbell (2020) describes advances in technology that lead to computers writing increasingly more code. The author asserts that while artificial intelligence may never fully replace human developers, it is becoming involved in a greater portion of the coding process as automation tools become more capable. The paper is a combination of history and an overview of bleeding-edge technologies (as of 2020), so it is more theoretical in nature, unlike this study, which aims to be practical.
- Ebert and Louridas (2023) look at generative AI and the benefits – along with the risks – of employing it in the software industry. The writers focus on the abstract concepts behind large language models and the features these provide to assist developers. The paper presents a summary of how generative AI works, how it can enhance productivity, and what precautions should be taken, but says little about what developers think about AI or how they are actually using it.
- Shi et al. (2023) consider how AI, mainly GitHub Copilot, helps developers write more secure code. The authors are largely optimistic, in contrast with Perry et al. (2023) and Vaidya and Asif (2023), who conclude that generative AI might end up introducing security vulnerabilities. Shi et al. (2023) focus more on how programmers can use generative AI when coding than on whether they would want to use such technology or how it could be improved.
- Tichy (1987) discusses AI tools, including natural language processors, and how these help developers write complex code more quickly and efficiently. Note that this article was written in 1987, so many of the concepts have already been implemented in the 3.5 decades since the paper's publication. Thus, the paper falls short of the goal of this study – namely, to examine current generative AI models and determine if they are useful to programmers.
- Vaidya and Asif (2023) warn that generative AI models like ChatGPT, being trained on a codebase that includes bad code as well as good, can produce highly insecure code even when prompted not to do so. The authors caution that unless programmers understand the code generated by AI, they may not be aware of the security vulnerabilities it contains. While

the paper addresses some shortfalls of generative AI, it does not inspect both sides so that readers can see whether AI makes a good coding assistant.

Based on the aforementioned articles, along with many others, artificial intelligence is not yet capable of replicating the skills of human developers. To ensure the code is secure, humans must thoroughly apprehend what it does and adjust it on their own rather than blindly trusting that the AI did a satisfactory job. However, in the hands of a programmer who knows these things and is ready to adapt and rewrite some or all of the code produced by AI, do tools like ChatGPT have the potential to increase productivity and accelerate the software development process? That is what this paper will consider.

METHODOLOGY

PARTICIPANTS

This study represents one researcher's experiment as part of a multi-case study. The participants were computer science students taking a class that was mandatory for their major. Each student completed a unique case, with a shared commonality of the same initial requirements while varying with customization of project design and AI assistant selection.

In total, there were ten college-aged researchers: nine males and one female. Each designed unique software for the client with the help of AI assistants. As the client (who was not digitally literate) only had need of the final working project, students selected coding languages they felt would be the most appropriate for coding and deploying the project to a remote client. Each student conducted research alongside the software development project, resulting in ten somewhat parallel research projects. Most of the researchers preferred Python, though all had varying levels of experience in other languages such as HTML/JavaScript, C++, and Java.

In addition to human participants, the AI chatbots were also participants in this study. Effectively, student researchers were Scrum masters and lead programmers, while chatbots were assistant coders that provided programming insights and contributed to the codebase. The human developers asked their virtual assistants questions regarding programming concepts and requested code samples, while the virtual assistants did their best to help, so to speak. All final coding decisions were left to the student researchers.

DATA COLLECTION METHODS

The project followed Agile methodologies designed for coding classes (see Bakke & Sakai, 2022) but adjusted to include a remote client. Throughout the course, as students designed their projects, the remote client provided initial requirements, gave feedback on the UI/UX flow designs, and participated in the final stages of testing. Since the client was often busy and because of scheduling conflicts, the client gave less feedback toward the middle of the project. During that time, the teacher offered additional feedback to ensure students still had an Agile experience. Due to the time constraints of a single semester, students were given the option to complete the project based on client feedback through the UI/UX design or to continue adjustments into the final project testing phase. Two students continued adjustments with the client, allowing full customization, and adjusted their design in the final weeks of the course, following Agile practices through the end of the class.

To determine the value of each AI assistant, students kept track of every time that AI was queried for help on the project. Students were allowed to query AI as often as they wished, but all queries were done with both assistants, allowing for AI-to-AI comparison at the end of the project. For example, if one AI were asked to assist with code for file retrieval, the other AI must be asked the same query. The results given by both chatbots could then be compared based on specific criteria. Criteria for AI assistant value were determined during the initial phases of the course, with each student selecting their own "AI as a coding assistant value" they wished to examine.

All data were collected in the context of developing a Find & Replace application for the remote client. As such, the students completed this research project over the course of a 15-week semester. All code generated by the AI chatbots contributed toward delivering an application suitable to the client's wishes; data collection stopped once the software was complete.

In this multi-case study, students independently designed their Find & Replace programs. At the same time, they used AI assistants in the coding process and collected data to address their research questions. Not all students chose the same AI chatbots or employed the same data analysis methods. The subsection below explains how data were gathered to address the research question: "Can ChatGPT and Bing Chat generate concise computer code better than a human developer could?"

Student case

When selecting a programming language for the Find & Replace application, the authors looked at several criteria. First, the program must be self-contained and not require the client to install third-party libraries. Second, the language should have good support for editing Office files (particularly .docx and .xlsx) and converting them to PDF files for export. It must not mess up the formatting of the documents when replacing text. Third, the language should have extensive documentation so the chatbots can easily generate code for a wide variety of tasks. Finally, the ideal language should have cross-platform support so the application can run on either desktop or mobile (desktop Windows being the main target) with minimal code changes. Speed was not among the criteria.

To gather data, the authors wrote a prompt that would be fed to ChatGPT and Bing Chat. Both chatbots were asked the same question, and their responses were recorded. Along with the code each chatbot generated, the collected data also consisted of the number of lines and characters in each response and any comments related to that test. These comments included whether one or both chatbots required multiple attempts to generate working code, whether the code necessitated major or minor modifications before it would run, and whether Bing Chat's "Precise" mode had to be used if the prompt exceeded the 2000-character limit for "Balanced" mode, among other things. Tests were conducted around once per week, representing milestones in the process of completing the Find & Replace project. The last test occurred shortly before the program was done and was ready to send to the client for testing and feedback. The authors chose to focus on quality over quantity, meaning fewer data points with extensive analysis instead of more data points with limited analysis of the results.

Note that the chosen metric – conciseness of code generated by AI – is only relevant when the AI is able to give working code, or at least code that can be made to work after multiple prompts and modifications. If the AI could not generate satisfactory code after several tries and major modifications, the code was not included in the dataset.

DATA ANALYSIS PROCEDURES

Data analysis involved two distinct categories of tests: quantitative and qualitative. For quantitative analysis, the author found the percentage of extraneous lines and characters in each chatbot's response by counting the total number of extraneous lines and characters and dividing the counts by the length of each response in lines and characters, respectively. The percentages were then averaged, and the result was compared to a threshold to determine whether the chatbots could produce concise computer code. If both averages (extraneous lines and extraneous characters) were under 2%, the chatbot would be deemed capable of writing concise code; if not, it would fail the quantitative test.

For qualitative analysis, the authors calculated the frequency of code requiring modifications, major or minor, as well as how often the chatbots failed to generate working code on the first try. These frequencies were then compared to thresholds: if a chatbot's code needed major modifications no more than 15% of the time and minor modifications no more than 30% of the time, and if no more than 30% of prompts required two or more tries to get working code, then the chatbot would pass qualitative analysis; if not, the chatbot would fail the test.

These percentages were chosen based on what an experienced developer might want from an AI assistant. If a chatbot produces concise, working computer code more than half the time (which is true if the frequencies are below the proposed thresholds – major modifications 15% of the time and minor modifications 30% of the time leaves 55% without needing modifications), it could be considered a useful assistant and not a waste of time. Since the optimal threshold depends on an individual's skill and coding speed, a 45% failure rate may be too high for many developers. However, it represents a bare minimum, which, if not met, disqualifies an AI assistant from being considered useful to most programmers until it is improved enough to meet the threshold.

DATA AND ANALYSIS

Initially, the authors tried to write the Find & Replace program in JavaScript. However, this did not turn out well since client-side JavaScript has limited abilities to edit Office files without paid third-party libraries. So the authors switched to C++ but ended up using Python (compiled for Windows with PyInstaller) because of its cross-platform nature and vast collection of libraries for working with Office files. Due to the sheer volume of Python code on public repositories and the extensive amount of Python documentation, the AI assistants (having been trained on this data) were more effective at generating code in Python than in JavaScript or C++.

According to the results of analyzing the data, both ChatGPT and Bing Chat passed the quantitative test for concise code. On average, code from ChatGPT had 0.7% extraneous lines and 1.4% extraneous characters, while code from Bing Chat had 0.7% extraneous lines and 1.1% extraneous characters. These totals were well below the 2% threshold, meaning that according to the authors' standard, both chatbots are able to produce concise computer code.

However, it was also found that neither ChatGPT nor Bing Chat passed qualitative analysis. Although both chatbots met the thresholds for how often it takes two or more tries to get a working answer – ChatGPT needed multiple tries 30% of the time, which just meets the threshold, while Bing Chat needed multiple tries 20% of the time – neither met the thresholds for producing code that runs without being modified first. While Bing's code only required major modifications 10% of the time, which was below the 15% threshold, it required minor modifications 70% of the time, far above the 30% threshold. ChatGPT failed to meet both thresholds, needing major modifications 30% of the time and minor ones 50% of the time (see Figure 1). In other words, only 20% of ChatGPT's code and 20% of Bing Chat's code worked without modifications.

	ChatGPT	Bing Chat
Extraneous lines (avg %):	0.7%	0.7%
Extraneous lines threshold:	2%	2%
Meets threshold?	Yes	Yes
Extraneous chars (avg %):	1.4%	1.1%
Extraneous chars threshold:	2%	2%
Meets threshold?	Yes	Yes
Took 2+ tries (% of times yes):	30.0%	20.0%
Took 2+ tries threshold:	30%	30%
Meets threshold?	Yes	Yes
Code needs major modifications (% of times):	30.0%	10.0%
Major modifications threshold:	15%	15%
Meets threshold?	No	Yes
Code needs minor modifications (% of times):	50.0%	70.0%
Minor modifications threshold:	30%	30%
Meets threshold?	No	No

Figure 1. The results of quantitative and qualitative analysis on ChatGPT (left column) and Bing Chat (right column)

It must be noted that the results varied significantly depending on which programming language was used. Both chatbots performed worse when the chosen language was C++ than when it was HTML/JavaScript or Python. This is likely because Python and HTML have more online documentation than C++. Also, C++ is more platform-dependent, so the chatbots may have mixed code samples designed to run on different platforms, resulting in a program that will not compile for any platform.

For a more detailed analysis of questions asked and how the chatbots performed, see Appendix A and Appendix B respectively. It contains two tables, which show the quantitative and qualitative analysis for each of the ten tests, which programming language each test used, and comments on how the numbers and ratings were chosen.

ETHICAL CONCERNS

Despite the limitations of generative AI, many developers still find it useful, citing increased productivity, faster debugging, and inspiration, among other reasons (see Campbell, 2020; Ebert & Louridas, 2023; Shi et al., 2023; Tichy, 1987). Nonetheless, the future of artificial intelligence in software development is uncertain due to current copyright lawsuits. According to Samuelson (2023), if the plaintiffs in such lawsuits triumph, generative AI may be restricted to code that is in the public domain or allows redistribution without citation, which would reduce AI's power to write code for a variety of contexts. Alternatively, it is possible that training the models on copyrighted data will remain legal while using the code they generate still violates copyright laws (Caballar, 2022).

Regardless of how these lawsuits turn out, it is advised that programmers who work with generative AI not integrate the suggested code unless they are willing to spend extra effort researching its sources. Developers could also refrain from asking for specific code and only request boilerplate code and ask questions on abstract programming concepts, avoiding the risk of introducing copyrighted code into the project. While neither of these solutions is optimal, they may be the wisest courses of action until these legal cases are settled.

Theoretically, one could build an AI chatbot that is aware of the sources of its code and able to cite them. Then, developers who employ the code could simply copy the citations into their own projects, eliminating the legal tension that arises from not knowing whether the code requires attribution for legal use. However, this would necessitate major rebuilding and retraining on the part of all major generative AI companies. Perhaps future breakthroughs in AI research will turn this vision into a reality.

CONCLUSION

Based on the results of this study, the first hypothesis (ChatGPT and Bing Chat cannot write as concise of code as a human developer) is incorrect since both chatbots met the threshold for concise code. The second hypothesis (Bing Chat will perform better than ChatGPT in generating concise code) is correct since Bing Chat outdid ChatGPT in all qualitative and quantitative tests, other than requiring minor modifications more often. However, an issue unforeseen when the hypotheses were proposed, namely the chatbots' inability to yield working code on the first try, made it impossible to fully test these hypotheses. (See the Limitations section for a more detailed discussion.)

The results of this study imply that generative AI can produce concise computer code, yet it struggles to write code that runs without modifications. The latter is arguably a more important metric than the former, as developers are more concerned that code works than with how concise it is. Thus, the results of this study are of limited importance given what programmers are currently looking for in a virtual assistant. However, as advances in AI enable it to generate code that contains fewer errors, developers will start to care more about quality and conciseness – so this study might be more relevant when that time comes.

There is still much improvement to be made in designing large language models that assist developers in writing computer code. Not only do even the best models fall short of producing code that is free of errors, but they also potentially violate copyright restrictions by not citing the sources of the code they memorize. Once these issues are addressed, though, AI is certainly capable of jumping the smaller hurdles of making its code more concise and efficient.

How can this study guide future AI software development? First, new models should be designed to decrease the error rate so that code runs without modifications more often. It appears that using more parameters and adding RAG results in better code with fewer errors. Second, more work can be done to improve results for programming languages with less documentation and publicly available code. Specifically, trainers could write more code examples by hand for languages with less documentation so that they perform equally well with languages that have more documentation. Third, generative AI would benefit from a mechanism that checks the code it produces. This could be a simple syntax analyzer or compiler that gives feedback to the AI, which would then fix the code iteratively until the result executes without errors. If implemented, these changes might well boost the accuracy of AI chatbots until they surpass the thresholds proposed in this paper.

Despite the challenges in this study, the authors were able to complete the application for the client, who was pleased with the result. Although this study presents a pessimistic outlook on generative AI's ability to help developers, the technology played an integral role in the development of the Find & Replace program for reasons beyond the scope of this study.

LIMITATIONS

The data collected for this study only includes questions related to three programming languages – HTML/JavaScript, C++, and Python – because the scope was limited to a single project. Consequently, while the dataset is more diverse than what questions focused on a single programming language could achieve, this study fails to account for the wide variety of languages developers employ worldwide.

Moreover, only ten samples were recorded, as data collection stopped when the Find & Replace project was finished. Although the analysis for each data point is highly detailed, the conclusions of this study are limited in strength due to the modest number of samples.

Finally, the chosen metric, whether the generated code is concise, is one part of a larger picture, which includes other metrics such as how often generative AI outputs working code, whether further prompting can improve the initial code, and how efficient the code is compared to a human developer's work. Only when artificial intelligence can give working code that is efficient and secure will conciseness be a significant factor. Once future generative AI models overcome the major hurdles existing ones face, such as code failing to run or containing security vulnerabilities, a future study may pick up where this one left off and reveal further insights into AI's ability to optimize the code it produces.

REFERENCES

Bakke, C., & Sakai, R. (2022). Using design-based research to layer career-like experiences onto software development courses. *Journal of Information Technology Education: Innovations in Practice*, *21*, 25–60. https://doi.org/10.28945/4988

Bankins, S., Ocampo, A. C., Marrone, M., Restubog, S. L. D., & Woo, S. E. (2024). A multilevel review of artificial intelligence in organizations: Implications for organizational behavior research and practice. *Journal of Organizational Behavior*, *45*(2), 159–182. https://doi.org/10.1002/job.2735

Caballar, R. D. (2022, November 19). *Ownership of AI-generated code hotly disputed: A copyright storm may be brewing for GitHub Copilot*. IEEE Spectrum. https://spectrum.ieee.org/ai-code-generation-ownership

Campbell, M. (2020). Automated coding: The quest to develop programs that write programs. *Computer*, *53*(2), 80–82. https://doi.org/10.1109/MC.2019.2957958

Ebert, C., & Louridas, P. (2023). Generative AI for software practitioners. *IEEE Software*, *40*(4), 30–38. https://doi.org/10.1109/MS.2023.3265877

Perry, N., Srivastava, M., Kumar, D., & Boneh, D. (2023). Do users write more insecure code with AI assistants? *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (pp. 2785–2799). Association for Computing Machinery. https://doi.org/10.1145/3576915.3623157

Samuelson, P. (2023). Generative AI meets copyright. *Science*, *381*(6654), 158–161. https://doi.org/10.1126/science.adi0656

Shi, Y., Sakib, N., Shahriar, H., Lo, D., Chi, H., & Qian, K. (2023, June). AI-assisted security: A step towards reimagining software development for a safer future. *Proceedings of the IEEE 47th Annual Computers, Software, and Applications Conference, Torino, Italy,* 991–992. https://doi.org/10.1109/COMPSAC57700.2023.00142

Tichy, W. F. (1987). What can software engineers learn from artificial intelligence? *Computer*, *20*(11), 43–54. https://doi.org/10.1109/MC.1987.1663415

Vaidya, J., & Asif, H. (2023). A critical look at AI-generate software: Coding with the new AI tools is both irresistible and dangerous. *IEEE Spectrum*, *60*(7), 34–39. https://doi.org/10.1109/MSPEC.2023.10177044

Zhou, X., Liang, P., Zhang, B., Li, Z., Ahmad, A., Shahin, M., & Waseem, M. (2023). *On the concerns of developers when using GitHub Copilot.* arXiv. https://doi.org/10.48550/arXiv.2311.01020

APPENDIX A

Here is an unedited example of the initial prompt fed to ChatGPT and Bing Chat, the responses from both chatbots, and the author's comments on the findings. This will give readers an idea of how the tests were conducted.

Here is an unedited example of the initial prompt fed to ChatGPT and Bing Chat, the responses from both chatbots, and the author's comments on the findings. This will give readers an idea of how the tests were conducted.

Prompt 1 (9/14/2023)

Please give me a JavaScript interface with these buttons: one called "UploadWordDocButton" with the text "Choose book" that when pressed gives a dialog to select a .docx file, one called "UploadExcelDocButton" with the text "Choose Excel sheet" that when pressed gives a dialog to select a .xlsx file, and one called "GoButton" with the text "GO!". GoButton should be grayed out until the .docx and .xlsx files have been chosen. Make the code as concise as possible.

ChatGPT's answer:

```
<html>
<head>
<script>
function enableGoButton() {
var docxFile = document.getElementById("UploadWordDocButton").value;
var xlsxFile = document.getElementById("UploadExcelDocButton").value;
if (docxFile && xlsxFile) {
document.getElementById("GoButton").disabled = false;
```

```
}
}
</script>
</head>
<body>
<input type="file" id="UploadWordDocButton" onchange="enableGoButton()" accept=".docx" />
<label for="UploadWordDocButton">Choose book</label>
<br />
<input type="file" id="UploadExcelDocButton" onchange="enableGoButton()" accept=".xlsx" />
<label for="UploadExcelDocButton">Choose Excel sheet</label>
<br />
<button id="GoButton" disabled>GO!</button>
</body>
</html>
```

Bing Chat's answer:

Here is a concise JavaScript interface with three buttons: "UploadWordDocButton", "UploadExcelDocButton", and "GoButton". The "UploadWordDocButton" and "UploadExcelDocButton" buttons will open a dialog box to select a .docx or .xlsx file, respectively. The "GoButton" will be grayed out until both the .docx and .xlsx files have been chosen.

```
const UploadWordDocButton = document.createElement("input");
UploadWordDocButton.type = "file";
UploadWordDocButton.accept = ".docx";
UploadWordDocButton.style.display = "none";
UploadWordDocButton.addEventListener("change", () => {
if (UploadExcelDocButton.files.length > 0) {
GoButton.disabled = false;
}
});

const UploadExcelDocButton = document.createElement("input");
UploadExcelDocButton.type = "file";
UploadExcelDocButton.accept = ".xlsx";
UploadExcelDocButton.style.display = "none";
UploadExcelDocButton.addEventListener("change", () => {

if (UploadWordDocButton.files.length > 0) {
GoButton.disabled = false;
}
});

const GoButton = document.createElement("button");
GoButton.textContent = "GO!";
GoButton.disabled = true;

document.body.appendChild(UploadWordDocButton);
document.body.appendChild(UploadExcelDocButton);
document.body.appendChild(GoButton);
```

I hope this helps!

Result:
ChatGPT: 22 lines, 746 characters
Bing Chat: 27 lines, 859 characters
Comments: ChatGPT's answer is better because it works out of the box, while Bing's is incomplete. I should have specified "HTML and JavaScript" in the prompt, not just JavaScript.

The full dataset with all ten tests is available on IPFS at this link:
ipfs://bafybeiew7ahtj6a52eslgxctdhzqgxpdqoqc5uzkq4xlktytrsidz7fbny/Testing Chatbots.docx

APPENDIX B

The following two tables show the detailed results from the qualitative and quantitative analysis tests:

ChatGPT:

Quantitative Analysis							*Qualitative Analysis*			
Test #	**Length of Code (lines)**	**Optimal Length (lines)**	**Extraneous Lines (%)**	**Length of Code (chars)**	**Optimal Length (chars)**	**Extraneous Chars (%)**	**Took 2+ Tries to Get Good Code?**	**Code Needs Modifications?**	**Programming Language**	**Comments**
1	22	22	0.0%	746	746	0.0%	No	None	HTML + JavaScript	None
2	19	19	0.0%	866	864	0.2%	No	Minor	HTML + JavaScript	The code won't run until I add the enableGoButton() function. I subtract 2 from "Optimal Length (chars)" since ChatGPT's code had 2 trailing spaces.
3	48	47	2.1%	1794	1753	2.3%	Yes	None	HTML + JavaScript	ChatGPT added .disabled class instead of styling based on "disabled" attribute, which would have made the code shorter.
4	93	92	1.1%	2143	2112	1.4%	No	Major	C++	ChatGPT used *main* instead of *WinMain*, so the file selection dialogs don't do anything. Also, the line "namespace fs = std::filesystem;" isn't necessary.
5	77	77	0.0%	831	831	0.0%	No	Major	C++	ChatGPT didn't explain how to run the code or what requirements it has (e.g. MiraPDF), so I couldn't use it.
6	547	547	0.0%	3429	3429	0.0%	Yes	Major	C++	Technically, the code is concise, but that's because it's only a skeleton with nothing implemented. It took a follow-up prompt to even get this far.
7	179	179	0.0%	2395	2395	0.0%	Yes	Minor	C++	The GUI isn't as beautiful as I'd hoped, but it does the job.
8	112	108	3.6%	1214	1101	9.3%	No	Minor	Python	Other than adding *suffix='.docx'* to the NamedTemporaryFile call so docx2pdf wouldn't throw an AssertionError, the code worked fine. (I was able to shorten ChatGPT's answer by 4 lines, so it wasn't 100% concise.)
9	342	342	0.0%	5579	5576	0.1%	No	Minor	Python	ChatGPT forgot to import *os* and *subprocess*, and the GUI doesn't have multiple pages as I wanted, but it does the job.
10	374	374	0.0%	6198	6185	0.2%	No	Minor	Python	ChatGPT's code requires some modifications since it fails to go to page 2 after "GO!" is pressed. Also, there's an unused QMessageBox import that should be removed.

Bing Chat:

Quantitative Analysis							*Qualitative Analysis*			
Test #	**Length of Code (lines)**	**Optimal Length (lines)**	**Extraneous Lines (%)**	**Length of Code (chars)**	**Optimal Length (chars)**	**Extraneous Chars (%)**	**Took 2+ Tries to Get Good Code?**	**Code Needs Modifications?**	**Programming Language**	**Comments**
1	27	25	7.4%	859	772	10.1%	No	Minor	HTML + JavaScript	Bing's code must be put in a window.onload() function before it does anything. It works fine except that it hides the choose file buttons for no reason, so the *style.display = "none"* lines should be removed.
2	27	27	0.0%	1224	1224	0.0%	No	Minor	HTML + JavaScript	Bing's code didn't do what I asked without modifications. The line *document.querySelectorAll("input[type=file]:valid").length === 2* was wrong.
3	47	47	0.0%	1728	1728	0.0%	Yes	None	HTML + JavaScript	Bing wouldn't give working code with "Balanced" mode, so I had to switch to "Precise" mode, which worked perfectly.
4	54	54	0.0%	1195	1195	0.0%	No	None	C++	Neither Bing nor ChatGPT did what I asked. Both produced command-line menus instead of full GUIs. But at least Bing's code worked on the first try.
5	63	63	0.0%	653	653	0.0%	No	Minor	C++	Since my prompt was longer than Bing's 2000-character limit for Balanced mode, I had to use Precise mode, which allows 4000 characters. Bing's code looks okay, but I didn't end up using it.
6	133	133	0.0%	1075	1071	0.4%	Yes	Major	C++	Again I used Precise mode since my prompt was over 2000 characters. It took a follow-up prompt for Bing to give me even boilerplate code.
7	208	208	0.0%	2377	2377	0.0%	No	Minor	C++	Bing's code has a bug where the buttons don't change color unless they are clicked twice. Otherwise, it's pretty good.
8	91	91	0.0%	1013	1013	0.0%	No	Minor	Python	Bing's code is very concise. Like ChatGPT, though, Bing didn't add *suffix='docx'* to the NamedTemporaryFile call, which is a minor fix.
9	235	235	0.0%	3317	3317	0.0%	No	Minor	Python	The code works perfectly fine (other than lacking the *subprocess* import). While the UI isn't as pretty as ChatGPT's, it does what I asked.
10	219	219	0.0%	3855	3842	0.3%	No	Minor	Python	Bing forgot to import *webbrowser* and imported *QHBoxLayout* unnecessarily. Otherwise, Bing did a great job – much better than ChatGPT.

The Excel sheet with that data is accessible at this IPFS link:

ipfs://bafybeiew7ahtj6a52eslgxctdhzqgxpdqoqc5uzkq4xlktytrsidz7fbny/Analyzing Chatbots.xlsx

AUTHORS

Andrew Millam (pronounced /ˈMaɪləm/) is an Honors graduate from Grand Canyon University in Phoenix, Arizona. In April 2024, he earned a Bachelor's degree in Computer Science, which included a six-month internship with Paychex as part of his capstone project. He is presently (as of July 2024) polishing his computer skills and engaging in training courses as he searches for a full-time job in software development or a related field.

Dr. Christine Bakke is an associate professor at Grand Canyon University, where she is an instructor in Cybersecurity, Information Technology, and Computer Science. After earning an IT PhD focused on active learning and educational robotics, she taught Computer Science, IT/Cyber, and Software Engineering for Universities in Northern Minnesota. Her work with students often incorporates prior experiences from her professional career working with networking, cybersecurity, databases, servers, and coding. Current research interests include custom coding projects, active learning, robotics, development of disability-assistive software, and IoT.