Evaluating the Impact of ChatGPT on Exercises of a Software Security Course

1st Jingyue Li
Dept. of Computer Science
Norwegian Univ. of Science and Technology (NTNU)
Trondheim, Norway
jingyue.li@ntnu.no

2nd Per Håkon Meland
Dept. of Computer Science
NTNU
Trondheim, Norway
per.hakon.meland@ntnu.no

3rd Jakob Svennevik Notland
Dept. of Computer Science
NTNU
Trondheim, Norway
jakob.notland@ntnu.no

4th André Storhaug
Dept. of Computer Science
NTNU
Trondheim, Norway
andre.storhaug@ntnu.no

5th Jostein Hjortland Tysse
Dept. of Computer Science
NTNU
Trondheim, Norway
jostein.h.tysse@ntnu.no

***Abstract*—Along with the development of large language models (LLMs), e.g., ChatGPT, many existing approaches and tools for software security are changing. It is, therefore, essential to understand how security-aware these models are and how these models impact software security practices and education. In exercises of a software security course at our university, we ask students to identify and fix vulnerabilities we insert in a web application using state-of-the-art tools. After ChatGPT, especially the GPT-4 version of the model, we want to know how the students can possibly use ChatGPT to complete the exercise tasks. We input the vulnerable code to ChatGPT and measure its accuracy in vulnerability identification and fixing. In addition, we investigated whether ChatGPT can provide a proper source of information to support its outputs. Results show that ChatGPT can identify 20 of the 28 vulnerabilities we inserted in the web application in a white-box setting, reported three false positives, and found four extra vulnerabilities beyond the ones we inserted. ChatGPT makes nine satisfactory penetration testing and fixing recommendations for the ten vulnerabilities we want students to fix and can often point to related sources of information.**

***Index Terms*—Software security, artificial intelligence, large language models, ChatGPT, IT education**

I. Introduction

Due to the risk and cost caused by software vulnerabilities, many universities have started educating software developers to develop secure code in the first place. The mandatory software security course for all IT students at our university teaches students how to incorporate security in each software development phase. Currently, the course focuses on security issues related to web applications. One critical module of the course is to teach students to understand the vulnerabilities listed in the open web application security project (OWASP) top 10 [1] and to identify and fix the vulnerabilities using state-of-the-art tools.

In the current course exercises, we teach students to use several tools to identify and fix vulnerabilities inserted in a web application. Students often use, e.g., Firefox web developer tool [2], Postman [3], and Zap [4], to gather web application information. Students also use scanners, e.g., Nessus [5], to scan the web application to identify vulnerabilities. In addition, students read the OWASP web security testing guide (WSTG) [6] to learn how to black-box test the web application and review the code to identify the vulnerabilities.

After ChatGPT [7], especially the GPT-4 model [8], is available on 14 March 2023, we suspect that students may use it to complete the exercise tasks without needing to learn other tools or read the testing guide [6]. Becker et al. [9] identified issues and opportunities in using AI-powered tools to teach an introductory programming course. Results of [10] showed that AI-powered tools could outperform students in answering exam questions. Tony et al. [11] found that chatbots could help identify vulnerabilities in the code, but with limitations. Studies, e.g., [12] and [13], showed that many of the codes generated from chatbots were vulnerable. However, to our knowledge, no study has investigated the impact of ChatGPT on teaching software vulnerability identification and fixing in a university course setup. In the coming years, we assume many students can access ChatGPT [7] or similar tools. Thus, we are motivated to understand ChatGPT's impact on teaching software security, especially its impact on the exercise design. For now, we focus on GPT-4 [8] because it allows for a context window of 8k tokens, meaning students can copy and paste a large chunk of code into it and let it identify and fix the vulnerabilities in the code for them. Our research questions are:

- RQ1: How accurate is ChatGPT in vulnerability identification?
- RQ2: How accurate is ChatGPT in recommending penetration test cases?
- RQ3: How accurate is ChatGPT in recommending vulnerability fixes?
- RQ4: How often can ChatGPT provide sufficient sources

978-1-6654-5223-6/23/$31.00

of information related to software vulnerability identification and fixing?

To answer the research questions, we asked teaching assistants of the course to input the vulnerable code to GPT-4 and use its prompts to guide vulnerability identification and fixing. We then calculated the identification and fixing recommendation accuracy of ChatGPT and summarized the fruitfulness of the source of information it provided.

The study results show that ChatGPT can provide satisfactory results to identify and fix the inserted vulnerabilities, although there are still rooms to improve. The contributions of the study are:

- We have identified the pros and cons of using ChatGPT to identify software vulnerabilities and propose fixing recommendations.
- To our knowledge, it is the first study investing in ChatGPT's impact on teaching software security, which gives valuable insight into updating the design of similar courses in the LLM era.
- Based on the results of this study, we provide visions for future research in software security and IT education.

The rest of the paper is organized as follows. Section II shows related work. Section III presents the background of the software security course. Section IV explains research design. The results are presented in Section V and are discussed in Section VI. Section VII concludes the study, and Section VIII presents future work.

II. Related work

An overview of the most recent studies relevant to our work is given in Table I.

TABLE I
OVERVIEW OF RELATED WORK

Ref.	Chatbot	Tests	Results
[11]	SKF	Performance, efficiency and user acceptance of chatbot vs. Internet search for a Java web application with XSS and SQL injection vulnerabilities.	Internet search outperformed SKF.
[10]	OpenAI Codex	Chatbot vs. students in solving programming questions.	Codex outperformed students.
[12]	OpenAI Codex	Comparison of Python code writing capabilities between Codex, GPT-3, and GPT-J.	Codex outperformed the others.
[14]	OpenAI Codex	Participants with chatbot support given programming tasks in Python, JavaScript, and C.	Participants wrote code with more vulnerabilities using chatbot than not using it.

Tony et al. [11] analyzed the effectiveness of the security knowledge framework (SKF) chatbot stemming from OWASP to find and fix security vulnerabilities by giving participants source code with known vulnerabilities. However, they only focused on Cross-Site Scripting (XSS) and SQL injection vulnerabilities. Their results showed that only three out of 15 participants arrived at the right fix with chatbot support, while seven out of 15 participants solved the task using a (manual) Internet search.

Finnie-Ansley et al. [10] showed that OpenAI's Codex outperformed 80% of students in exam questions related to the introductory programming course, which could provide low-risk/high rewards for students focusing on getting good grades rather than developing an understanding, leading to academic misconduct.

Chen et al. [12] analyzed the Python code-writing capabilities of OpenAI's Codex and showed that OpenAI's Codex could produce vulnerable or misaligned code. Santhanam et al. [15] found that around 2/3 of the chatbots for code generation used Stack Overflow as their main information source. Fischer et al. [13] figured out that out of 1.3 million Android applications using code snippets from Stack Overflow, 97.9% contained at least one insecure code snippet.

Perry et al. [14] examined how developers chose to interact with AI code assistants and how those interactions caused security mistakes. They found that their participants with chatbot access wrote less secure code than those without such access, despite that the participants themselves thought it would be vice-versa. This showed that such tools could give users a false sense of the security of the code.

III. Course background

Although the detailed requirements and grading criteria of the mandatory exercises of the software security course vary each year, the exercises are usually designed as follows.

- We develop a new web application and insert around 20 to 30 vulnerabilities listed in the OWASP top 10 [1] each year. The web application in the Spring 2023 semester simulates functionality for refugees to find volunteer help services. Certified volunteers register their wishes and skills to provide help services through the web application. Administrators of the web application handle skill certifications to verify the volunteers. The web application uses Django 4.0.8 with Django Rest Framework 3.13.1 as the back end. The front-end code is developed using React 4.0.3. To route the requests to the front end and back end, we use NGINX 1.23.1 as a reverse proxy. The web application is hosted at the internal Gitlab repository of our university and is only accessible with students' university credentials. Thus, we believe the vulnerable web application used in 2023 is not included in ChatGPT's training dataset.
- In the first module of the exercise, students must first use tools to collect the web application's information and possibly make a page map. Based on the page map, students must identify the inserted vulnerabilities using code reviews and black-box testing approaches and tools. Students need to write reports to describe how the tests are performed to identify the vulnerabilities and the location of the vulnerable codes. Besides reporting the vulnerabilities and their locations, students must also provide vulnerabilities' WSTG [6] code to refer the vulnerabilities to proper categories.

- In the exercise's second module, students must fix the identified vulnerabilities. To reduce the students' effort to fix the vulnerabilities, we do not ask them to fix all vulnerabilities. We make a list of, for example, ten vulnerabilities and let students fix only those. By limiting and predefining the set of vulnerabilities to identify and fix, we make it easier to make a solutions guide and coach our teaching assistants.

As more than 200 students take this course each year, the exercises are often organized in groups containing one to three students. We give grades to students based on reading their reports and counting the number of vulnerabilities they identify and fix correctly. We do not punish students if they report false positive vulnerability identification results. Sometimes, students find and report more vulnerabilities than we have inserted. If students identify extra vulnerabilities, we give them bonus points up to the full score of the exercises. In parallel with these exercises, we also offer students free access to a software game, Secure Code Warrior [16], which teaches software security, as a supplement. An example feature of the Secure Code Warrior game is that it provides multiple-choice questions and asks students to identify a location of code that is vulnerable among multiple code locations.

In previous years, the grades of the exercises were added to the course's final grades. From the Spring 2023 semester, the exercises grade are only used to qualify a student to take the final written exam if the student's exercise grade passes a threshold. The final written exam counts as 100% of the course grade.

IV. RESEARCH DESIGN

This section explains the detailed design to answer each research question. In general, we designed the prompts for ChatGPT in the same way that the exercises are structured and how we expect students to approach them. We standardized the prompts such that we first asked our question before pasting the relevant code. We asked follow-up questions until we could verify whether ChatGPT could provide a satisfactory answer.

A. Design and implementation to answer RQ1

To answer RQ1, one teaching assistant, who had inserted the vulnerability in the web application as the preparation of the exercises, evaluated how accurately ChatGPT identified vulnerabilities in the white-box settings. The teaching assistant first copied and pasted the code of each front- and back-end file into ChapGPT and asked it to identify the vulnerabilities in the code. As each file contains fewer than 8k tokens, the teaching assistant did not need to split the codes in the file when inputting them into ChatGPT. An example prompt the teaching assistant used was as follows.

This is the settings.py for a Django project. Can you find any vulnerabilities and list them according to OWASP top 10 or with the OWASP WSTG id? Also tell me explicitly which line of code is the problem.

In addition, the teaching assistant asked ChatGPT to provide the WSTG [6] code as follows.

Do you have the owasp wstg code for these vulnerabilities?

Sometimes, ChatGPT did not give a complete list of vulnerabilities in the first place. So, the teaching assistant asked an additional question as follows.

Do you find any other vulnerabilities in the file? Do you find anything else?

As students often use the WSTG [6] as a checklist to identify vulnerabilities, the teaching assistant also simulated such a process and asked more concrete questions as follows to identify a particular type of vulnerability in the code:

How about the token lifetimes, are they sufficient?

After applying similar prompts on each file, the teaching assistant calculated the accuracy of the ChatGPT's vulnerability identification.

B. Design and implementation to answer RQ2

As aforementioned, we specified only ten vulnerabilities for students to fix. To answer RQ2, we decided to focus on asking ChatGPT to propose penetration test cases only on those ten vulnerabilities to limit the teaching assistant's effort in performing this study. Another teaching assistant, who assisted in the vulnerability insertion, asked ChatGPT to propose penetration test case to answer RQ2. The example prompts for penetration testing are as follows:

How can I exploit the following code by XSS?

Show me an example html injection.

How can I observe unencrypted HTTP network traffic from my computer?

C. Design and implementation to answer RQ3

The teaching assistant, who answered RQ2, also performed the study to answer RQ3. The example prompts are as follows to let ChatGPT first identify the vulnerability through penetration testing and then propose fixes.

Does my login function below have any lockout functionality?

Then, the teaching assistant asked:

How can I test whether my application has lockout functionality in a black-box fashion?

At last, the teaching assistant asked:

Show me how to use Django-axes to solve this problem.

For each of the ten vulnerabilities, the teaching assistant used similar prompts and counted the accuracy of the proposed penetration test cases by ChatGPT to answer RQ2 and fixing solutions proposed by ChatGPT to answer RQ3.

D. Design and implementation to answer RQ4

To answer RQ4, a third teaching assistant first asked ChatGPT to identify security vulnerabilities in each file using prompts as follows.

I want you to identify security vulnerabilities in the following file plus the code with vulnerabilities.

The ChatGPT usually supplies a response with the results. The teaching assistant then asked:

Can you provide sources for that information?

The teaching assistant did not include any notion of Django, OWASP, etc. This is to ensure the results are not biased towards any particular source. After asking ChatGPT to identify security vulnerabilities in two front-end files, three back-end files, one Django setting file, and one NGINX configuration file, the teaching assistant counted the number of cases in which ChatGPT provided sufficient source of information.

V. Research results

A. Results of RQ1

Results of vulnerability identification in the white-box settings showed that GPT-4 found 16 of our 28 listed vulnerabilities on the first try. With some additional more specific prompts, four more were found. The ones identified by ChatGPT include, for example, sensitive information sent via unencrypted channels, default admin password, and unlimited login attempts without a lockout. A complete list of the identified vulnerabilities is in the file [1].

The eight vulnerabilities that were not found included, for example, unsafe user registration process (multiple users can have the same email), username enumeration with error messages from user registration, and reset password does not validate token correctly. A complete list of the missed vulnerabilities is also in the file [1]. A possible explanation of the missed vulnerabilities could be that ChatGPT did not fully understand the code's behavior or did not access every line of the code in one prompt. The built-in Django methods and imported codes caused several of these vulnerabilities. However, in some cases, as with "reset password does not validate", ChatGPT totally missed a logical error where the random token was not used to validate the reset password link.

ChatGPT reported three false positives, namely, improper error handling, insecure direct object reference in the document download method, and broken access control – cross-origin resource sharing (CORS) settings. The allowed hosts' CORS settings were reported as a vulnerability by ChatGPT. However, in our configuration to run the application on the server, it is not a vulnerability. If students blindly trust the results from ChatGPT, false positives may confuse them since they cannot perform the exploits for the report.

Beyond the vulnerabilities we inserted, ChatGPT identified four extra ones, namely, admins able to see more information than they should, no CAPTCHA for registration, SQLite used in production, and improper exception handling – broad exception catching.

To get a full score on the exercise, we required the students to find 15 vulnerabilities. The results of RQ1 mean that ChatGPT can pass the threshold.

B. Results of RQ2

ChatGPT proposed satisfactory methods to penetration test the vulnerabilities in nine of the ten cases. Data in the file [2] shows an overview of the tests. ChatGPT struggled with answering our questions when:

1) It has ethical issues showing penetration test cases.
2) The number of files and code length exceeds the input limit.
3) ChatGPT might assume that our application has some default components. On the other hand, it overlooked details in our code.

The combination of the issues caused the flawed answer from ChatGPT. The flawed penetration test is about SQL injection. ChatGPT proposed the following:

```
request_id = "1' UNION SELECT * FROM auth_user WHERE
    username = 'admin' AND '1' = '1"
```

This is an injection, and it will execute in the database. However, there is an issue that the auth_user table is not part of our models. We have extended Djangos default models and named the table users_user instead. Therefore the injection will cause an error, and it does not extract any useful information or does any harm to the database. When we provide ChatGPT with more context, it will stop providing injections because of ethical issues:

Please note that providing working SQL injection examples might encourage malicious activities, which is against OpenAI's policy to promote such behavior.

C. Results of RQ3

As shown in the file [2], ChatGPT provided satisfactory recommendations for fixing the vulnerabilities in nine of the ten cases. Regarding the flawed mitigation recommendation of "sensitive information sent over unencrypted channels", ChatGPT struggled to provide a complete answer. We provided seven files relevant to configuring HTTPS:

- .env: provides PORT_PREFIX, GROUP_ID, DOMAIN, PROTOCOL (PORT_PREFIX and GROUP_ID compose the server port).
- Dockerfile (front end): Composes the .env variables into an URL to send API requests.
- Dockerfile (back end): Passes the .env variables to Django.
- settings.py: composes the .env variables for user redirects.
- Dockerfile (nginx): Passes nginx.conf.
- nginx.conf: Configures the reverse proxy.
- docker-compose.yml: Builds the project using the Dockerfiles and defines port forwarding.

ChatGPT suggested a viable nginx.conf and showed how to generate the certificate/key. However, it suggested a flawed port configuration as below for docker-compose.yml, where two ports were forwarded to the same port, causing an error:

```
ports:
  - ${PORT_PREFIX}${GROUP_ID}:80
  - ${PORT_PREFIX}${GROUP_ID}:443
```

By adjusting the nginx.conf file to incorporate the HTTPS configuration suggested by ChatGPT and simultaneously limiting port forwarding to only port 443, we caused CORS errors

[1] https://doi.org/10.6084/m9.figshare.23576370

[2] https://doi.org/10.6084/m9.figshare.23628879

to emerge from Django. The root cause was that ChatGPT did not acknowledge the critical role of the PROTOCOL variable in the .env file. This variable delineates whether the URL would employ HTTP or HTTPS when making API calls and functions as a mechanism to establish the list of allowed origins in Django. When ChatGPT recommended the implementation of HTTPS via redirection of HTTP requests, as opposed to direct utilization of HTTPS, it created a conflict between the incoming request and the expected allowed origin.

The new HTTPS endpoint was intended to replace the previously forwarded HTTP endpoint using the same port. By guiding the students to modify the PROTOCOL to HTTPS and use a pure HTTPS endpoint, we had already circumvented the issues introduced by ChatGPT. This instance exemplified ChatGPT's challenges in fully grasping the context and its intricate interdependencies.

D. Results of RQ4

The ChatGPT provided sufficient sources of information in all cases and pointed to different resources for different vulnerabilities as follows.

- For vulnerabilities in the settings file, ChatGPT provided references to the Django documentation and some GitHub repositories.
- For back-end-related vulnerabilities, ChatGPT provided mostly references to the Django documentation and OWASP top 10 [1].
- For front-end-related vulnerabilities, ChatGPT provided mostly references to OWASP top 10 [1] and MDN (Mozilla Developer Network).
- For NGINX-related vulnerabilities, ChatGPT provided mostly references to MDN (Mozilla Developer Network) and the NGINX documentation.

In summary, ChatGPT could provide relatively good sources. All the suggested sources were relevant websites. For example, it referenced the Django security documentation and OWASP top 10 [1]. On occasions, it even provided Stack Overflow references. However, some of the sources are a bit outdated. For example, the OWASP top 10 is from 2017.

However, when the teaching assistant asked:

Where did you get that information from?

The answers from ChatGPT were more generic, such as:

As an AI language model, my knowledge is based on the vast amount of text data that I have been trained on, which includes information from a variety of sources such as websites, books, articles, and other documents. Specifically, for Django-related information, I rely on the official Django documentation and various articles, tutorials, and blog posts written by experienced Django developers and experts.

My training data includes text sources up to September 2021. It is important to note that my knowledge does not include the latest updates or developments that may have occurred after this date. For the most up-to-date information, it's always a good idea to consult the official Django documentation or other trusted sources.

VI. Discussion

A. Comparison with related work

Compared to the study [11] using the SKF chatbot for security identification, our study covers many more types of vulnerabilities than Cross-Site Scripting and SQL injection. Our study shows that ChatGPT can precisely identify many types of vulnerabilities in large file chunks but is still limited in identifying cross-file vulnerabilities across the codebase.

Regarding [9] focusing on the impact of AI-based approach on software engineering education, our results raise a similar alarm, i.e., course exercise design and grading shall be updated in the LLM era. However, our study investigates the issue from identifying and fixing software vulnerability perspectives rather than automatically generating and completing functional code.

The results of [12], [14] showed that the code generated from LLMs might be vulnerable. We applied ChatGPT to identify vulnerabilities and found that ChatGPT could miss identifying a few types of vulnerabilities, indicating that vulnerable code in ChatGPT's training dataset might also degrade its capability to detect vulnerabilities.

B. Implication

Results of RQ1 indicate ChatGPT excels at detecting vulnerabilities within its context window, aligning with OpenAI's findings [17]. However, detecting complex vulnerabilities across multiple files can be challenging due to its 8k token limit, despite being double that of similar models. This limit restricts the codebase analyzed, requiring knowledge of which files to assess together for cross-file vulnerabilities. Access to the 32k version of GPT-4 could enhance detection capabilities by allowing larger codebase analysis, thereby streamlining the detection process.

Results of RQ2 and RQ3 show ethical concerns arising from penetration testing questions. Simultaneously, the input limit affects the context we can provide for ChatGPT, potentially leading to false assumptions or overlooked code characteristics. Results of RQ4 demonstrate ChatGPT's ability to provide students with adequate information to support their solutions.

This study suggests that the introduction of ChatGPT presents teaching challenges in software security. If we continue grading based on counted vulnerabilities found, exploited, and fixed, students may score well by leveraging ChatGPT. We could consider blocking access to ChatGPT during exercises, but this could prove difficult to enforce, given that students complete the exerciseses mainly from home.

We could revise our exercise grading system to include report writing, where students compare various tools and solutions. However, this would disrupt our existing quantitative approach, requiring new procedures to ensure fair and consistent evaluations by the teaching assistants. This is crucial, especially if exercise grades impact course grades directly. Incorporating qualitative assessments, like discussions, would require a framework to maintain fairness and consistency. Moreover, the possibility of students leveraging AI to answer discussion questions should also be considered.

Our study reveals ChatGPT's difficulty in detecting cross-file vulnerabilities in codebases. While incorporating more such vulnerabilities could increase the challenge of using chatbots for tasks, this may make some vulnerabilities less realistic, as real ones often are contained within a single file. Should we require students to locate cross-file vulnerabilities, we may need to frame the exercise questions to hint at their location.

ChatGPT can enhance students' comprehension of software security issues. We currently introduce penetration testing tools and automated scanners, stressing their limitations as per OWASP's testing guide [18]. We can treat ChatGPT similarly and teach students its use alongside its limitations.

Today, students must find 15 of 28 inserted vulnerabilities to get a passing grade for the exercise due to the limited time they can spend on the course. In the future, by teaching students to use ChatGPT to identify and fix vulnerabilities, we can raise the threshold to require students to find more vulnerabilities. Thus, we can motivate them to understand more types of vulnerabilities.

C. Threat to validity

The teaching assistants and course teachers first agreed upon the study design to avoid data analysis biases. The data analysis results were shared among the teaching assistants using the functions of https://sharegpt.com/ and were cross-checked. Although the study focused on only one software security course and one web application, we believe the insights from this study can be generalized to other software security courses focusing on web application security. The web application we focused on includes typical features, such as registration, login, password encryption, session management, etc., related to web application security and popular OWASP top 10 vulnerabilities.

VII. Conclusions

This study piloted ChatGPT to identify its possible impacts on teaching a software security course, especially on the exercises of the course. The results show that students can easily get good exercise grades by simply asking ChatGPT and copying the results to the reports. The results also provide valuable insights into the advantages and limitations of using ChatGPT for identifying and fixing typical web application security issues.

VIII. Future work

There is a wide range of research tasks that can be envisioned in the field of chatbot development and secure coding. In the coming year, we plan to provide a selected group of students access to ChatGPT, allow them to do exercises using it, and then interview them to identify the pros and cons of ChatGPT from students' perspectives. We encourage the community to perform similar studies to establish more empirical evidence, for instance, identifying local variations, performance for different programming languages and trends over time. We also foresee studies where we can evaluate and compare large language models from different organizations as they continue to become available. A challenge here is that the technology and datasets develop so rapidly that results from most benchmark experiments are quickly outdated. A framework for structuring the research and maybe even automating experiments would be useful assets to the research community.

Data availability

The source code of the web application is available at https://doi.org/10.6084/m9.figshare.23576364. All the prompts and ChatGPT results related to this study are available at https://doi.org/10.6084/m9.figshare.23629455.

Acknowledgment

The research conducted in this paper has partly been related to the CyberSecPro project under the European Union's Digital Europe Programme (DEP) (grant agreement No 101083594).

References

[1] OWASP, "Owasp top ten," 2023. [Online]. Available: https://owasp.org/www-project-top-ten/
[2] Fireforx, "Firefox web developer tool." [Online]. Available: https://firefox-dev.tools/
[3] Postman, "Postman," 2023. [Online]. Available: https://www.postman.com/
[4] OWASP, "Owasp zed attack proxy." [Online]. Available: https://owasp.org/www-project-zap/
[5] Tenable, "Nessus vulnerability assessment." [Online]. Available: https://www.tenable.com/products/nessus
[6] OWASP, "Web security testing guide 4.2," 2023. [Online]. Available: https://owasp.org/www-project-web-security-testing-guide/v42/
[7] OpenAI, "Chatgpt," 2023. [Online]. Available: https://openai.com/blog/chatgpt
[8] ——, "Gpt4," 2023. [Online]. Available: https://openai.com/product/gpt-4
[9] B. A. Becker, P. Denny, J. Finnie-Ansley, A. Luxton-Reilly, J. Prather, and E. A. Santos, "Programming is hard–or at least it used to be: Educational opportunities and challenges of ai code generation," *arXiv preprint arXiv:2212.01020*, 2022.
[10] J. Finnie-Ansley, P. Denny, B. A. Becker, A. Luxton-Reilly, and J. Prather, "The robots are coming: Exploring the implications of openai codex on introductory programming," in *Proceedings of the 24th Australasian Computing Education Conference*, ser. ACE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 10–19. [Online]. Available: https://doi.org/10.1145/3511861.3511863
[11] C. Tony, M. Balasubramanian, N. E. Díaz Ferreyra, and R. Scandariato, "Conversational devbots for secure programming: An empirical study on skf chatbot," in *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*, 2022, pp. 276–281.
[12] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.
[13] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack overflow considered harmful? the impact of copy&paste on android application security," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 121–136.
[14] N. Perry, M. Srivastava, D. Kumar, and D. Boneh, "Do users write more insecure code with ai assistants?" *arXiv preprint arXiv:2211.03622*, 2022.
[15] S. Santhanam, T. Hecking, A. Schreiber, and S. Wagner, "Bots in software engineering: a systematic mapping study," *PeerJ Computer Science*, vol. 8, p. e866, 2022.
[16] Secure Code Warrior, "Secure code warrior." [Online]. Available: https://www.securecodewarrior.com/
[17] OpenAI, "Gpt-4 technical report," 2023.
[18] M. Meucci and A. Muller, "Owasp testing guide, v4," *OWASP Foundation*, vol. 4, pp. 14–23, 2014.