AI Writes, We Analyze: The ChatGPT Python Code Saga

Md Fazle Rabbi, Arifa Champa, Minhaz Zibran
Idaho State University, USA
{mdfazlerabbi,arifaislamchampa,minhazzibran}@isu.edu

Md Rakibul Islam
Lamar University, USA
mislam108@lamar.edu

ABSTRACT

In this study, we quantitatively analyze 1,756 AI-written Python code snippets in the DevGPT dataset and evaluate them for quality and security issues. We systematically distinguish the code snippets as either generated by ChatGPT from scratch (ChatGPT-generated) or modified user-provided code (ChatGPT-modified). The results reveal that ChatGPT-modified code more frequently displays quality issues compared to ChatGPT-generated code. The findings provide insights into the inherent limitations of AI-written code and emphasize the need for scrutiny before integrating such pieces of code into software systems.

KEYWORDS

Code quality, Code security, ChatGPT, Python, CWE, Analysis

ACM Reference Format:
Md Fazle Rabbi, Arifa Champa, Minhaz Zibran and Md Rakibul Islam. 2024. AI Writes, We Analyze: The ChatGPT Python Code Saga. In *21st International Conference on Mining Software Repositories (MSR '24), April 15–16, 2024, Lisbon, Portugal.*

1 INTRODUCTION

The field of software engineering has changed a lot recently as conversational Artificial Intelligence (AI) like ChatGPT has become popular. Developers and non-developers now use these AI systems to seek help with all kinds of tasks. Among the many ways ChatGPT is used, programming assistance makes up a major category.

According to a survey [37] in 2023, 92% of US developers say AI tools like ChatGPT help them be more productive at programming tasks. This growing reliance on AI is not surprising. With its ability to generate content, ChatGPT has proven itself as a great coding assistant - it can quickly help developers implement algorithms and solve problems. Validating this, a study [4] by the National Bureau of Economic Research highlighted the potential of generative AI such as ChatGPT to boost workforce productivity by 14%.

With new technology comes potential issues. While ChatGPT can provide executable code, there are growing concerns about the quality and security of the code it generates [16]. Using poor quality or insecure ChatGPT code in projects could cause anything from small bugs to huge security breaches resulting in a compromise of the entire system.

MSR '24, April 15–16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0587-8/24/04…$15.00
https://doi.org/10.1145/3643991.3645076

In this work, we examine the quality and security of Python code snippets produced through interactions of developers with ChatGPT. In particular, we address two research questions (RQs):
RQ1: How is the quality of Python code developed with assistance from ChatGPT?
RQ2: How prevalent are different security vulnerabilities in ChatGPT-aided Python code?

A good understanding of the prevalent security and quality issues in source code generated/aided by ChatGPT is beneficial in two ways. First, the developers will be informed of such issues so that they can particularly examine in search for such issues and sanitize before using the AI-generated code in their projects. Second, AI systems like ChatGPT can be better trained around those issues leading to more trustworthy AI-tools capable of generating more reliable source code. Therefore, we address the aforementioned research questions by carefully examining 1,756 Python code snippets produced through developers' interactions with ChatGPT.

2 METHODOLOGY

2.1 Dataset

The 1,756 code snippets and corresponding interactions between ChatGPT and the developers are drawn from six different sources in the DevGPT dataset [41]. We use the DevGPT snapshot, labeled '20230914', obtained on September 15, 2023. The snapshot includes 17,622 prompts and ChatGPT responses including 12,031 code snippets among which 1,756 are written in Python.

We particularly focus on Python code for two reasons. First, Python has been one of the three most popular programming languages worldwide for nine consecutive years [15]. Second, Python code is also dominant in the chosen snapshot of the DevGPT dataset. Out of the total 12,031 code snippets, 1,756 are written in Python, followed by 1,638 bash code and 1,500 JavaScript snippets.

2.2 Approach

First, we extract Python code snippets provided by ChatGPT in its responses resulting in 1,756 Python snippets. Then we identify prompts from the conversations that contain Python code in the responses and we extract any Python code snippets contained within these prompts. This task is accomplished by leveraging the `gpt-3.5-turbo` API [31].

This process results in two distinct sets of code snippets for each ChatGPT conversation: one containing code from the prompts that are *developer-provided* and another containing code snippets in the responses from ChatGPT. We observe that without any developer-provided code in the prompts, ChatGPT generates completely new code, which we call *ChatGPT-generated*. When prompts contain developer-provided code, ChatGPT may slightly modify the provided code, which we call *ChatGPT-modifed*. Thus, there remains a substantial similarity between a developer-provided code snippet and the corresponding *ChatGPT-modifed* code snippet.

Hence, to distinguish the *ChatGPT-modifed* code snippets, we measure similarities between the code snippets from prompts and responses using cosine similarity [28]. We test with cosine similarity thresholds from 0.2 to 0.8 on 50 random code pairs and settle on a threshold of 0.7. A similar approach was adopted by Wu et al. [40]. Thus we distinguish 213 python snippets as ChatGPT-modified. The rest 1,543 are regarded as ChatGPT-generated. We continue with comparative analyses between *ChatGPT-modifed* and *ChatGPT-generated* code. For the 213 ChatGPT-modified code snippets, we also preserve the corresponding 213 developer-provided code snippets from the original prompts.

2.2.1 Measuring Code Quality. Using Pylint 3.0.2 [35], we separately analyze the *ChatGPT-modifed* and *ChatGPT-generated* code snippets to capture key code quality issues of four *types*: errors (E), conventions (C), warnings (W), and refactoring (R). We exclude `import`-related issues from our analysis due to Pylint's limited reliability in accurately assessing `import` statements [39]. We also overlook missing docstring-related issues, i.e., C0114 (missing module), C0115 (missing class), and C0116 (missing function) as ChatGPT-aided code might not always include docstrings. Additionally, we discard style-related issues such as whitespace, newlines, and invalid naming conventions.

2.2.2 Capturing Security Vulnerabilities: Using Bandit 1.7.5 [34], we again separately analyze the *ChatGPT-modifed* and *ChatGPT-generated* code snippets for capturing security vulnerabilities in them. Upon analysis, Bandit provides a report detailing potential issues of different *types* having each *type* identified with a CWE (Common Weakness Enumeration) ID [8]. For each type of issue identified as a CWE, Bandit generates one or more issues detailing the specific security vulnerabilities in deeper detail.

2.3 Metrics

For the comparative quantitative analysis, we define the following four metrics concerning issue types (denoted as τ) and individual issues (denoted as i). For code quality issues, $\tau \in \{E, C, W, R\}$. For security issues, $\tau \in \{\text{all CWEs}\}$.

- For each issue *type* τ, the average number of issues of type τ per code snippet, denoted by α_τ, is calculated as:
$$\alpha_\tau = \frac{\text{Total occurrances of issues of type } \tau}{\text{Total number of code snippets}} \quad (1)$$
- For each issue *type* τ, the proportion of code snippets with at least one issue of type τ, denoted as β_τ, is obtained by:
$$\beta_\tau = \frac{\text{Total code snippets with issues of type } \tau}{\text{Total number of code snippets}} \quad (2)$$
- For each issue i, the average number of occurrences of issue i across snippets, denoted as γ_i, is computed as:
$$\gamma_i = \frac{\text{Total number of occurrences of issue } i}{\text{Total number of code snippets}} \quad (3)$$
- For each issue i, the proportion of code snippets having one or more occurrences of issue i, denoted as ν_i, is defined as:
$$\nu_i = \frac{\text{Total code snippets having the issue } i}{\text{Total number of code snippets}} \quad (4)$$

3 ANALYSIS AND FINDINGS

For the *ChatGPT-generated* code snippets, we compute all the aforementioned metrics separately for code quality issues and security issues. We do the same for *ChatGPT-modified* code snippets as well.

3.1 Code Quality Assessment

For ChatGPT-generated and ChatGPT-modified code snippets, in Table 1, we summarize α_τ and β_τ metric values concerning the four types of code quality issues. As seen in the table, the values of both these metrics are higher for ChatGPT-modified code across all four types of issues. We find that 76.46% (i.e., 1180 out of 1543) of ChatGPT-generated code snippets have one or more code quality issues whereas the proportion is 84.51% (i.e., 180 of 213) for the ChatGPT-modified code. This implies that ChatGPT-modified code is more prone to quality issues compared to ChatGPT-generated code.

Table 1: Summary of code quality issue types

Type (τ)	In ChatGPT-generated Code		In ChatGPT-modified Code	
	α_τ	β_τ	α_τ	β_τ
E	1.70	0.55	3.08	0.54
C	0.27	0.13	1.31	0.42
W	0.53	0.24	0.96	0.31
R	0.18	0.13	0.26	0.21

To get an idea of whether or not the issues in ChatGPT-modified code are introduced by ChatGPT or they might have already existed in the developer-provided code, we make a comparison between developer-provided and ChatGPT-modified code.

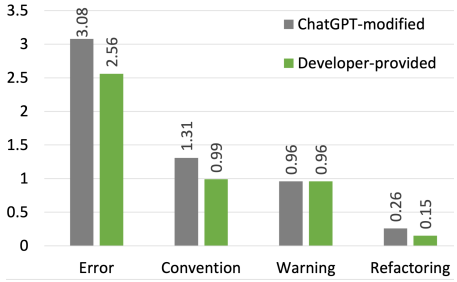

Figure 1: Code quality issues per code snippet (α_τ)

Figure 1 shows that developer-provided code has lower rates of errors, convention violations, and refactoring suggestions (with equal rates of warnings) compared to ChatGPT-modified code. This indicates the possibility that ChatGPT's modifications might have introduced new issues to the developer-provided code.

3.1.1 Most Frequent Quality Issues. As seen in Table 1, errors are the most frequent issues in both ChatGPT-generated and ChatGPT-modified code, while refactoring suggestions are the least frequent.

From our analysis, we find that all the 1,543 ChatGPT-generated code snippets collectively contain 4,156 occurrences of 69 distinct issues. In contrast, the 213 ChatGPT-modified code snippets collectively have 1,196 occurrences of 38 distinct issues. We compute γ_i and ν_i metric values for each of the 69 district issues found in ChatGPT-generated code and for each of the 38 distinct issues in ChatGPT-modified code. Due to limitation of space, in Table 2, we present results for the five most frequent issues in each issue type found in ChatGPT-modified and ChatGPT-generated code.

As seen in Table 2, in each type of code quality issues, two or more most frequent issues are common, as highlighted and marked in italics, in both ChatGPT-modified and ChatGPT-generated code. For example, E0602, E0001, and E1101 are among the five most

Table 2: Most frequent code quality issues in ChatGPT-generated and ChatGPT-modified code snippets

Type (τ)	In ChatGPT-generated Code			In ChatGPT-modified Code		
	Issue	γ_i	ν_i	**Issue**	γ_i	ν_i
Error (E)	*E0602:undefined-variable*	1.475	0.409	*E0602-undefined-variable*	2.939	0.502
	E0001-syntax-error	0.099	0.099	*E1101-no-member*	0.080	0.038
	E1101-no-member	0.084	0.029	E1120-no-value-for-parameter	0.019	0.019
	E0611-no-name-in-module	0.014	0.011	*E0001-syntax-error*	0.014	0.014
	E0104-return-outside-function	0.008	0.007	E0601-used-before-assignment	0.009	0.009
Convention (C)	*C0301-line-too-long*	0.251	0.126	*C0301-line-too-long*	1.249	0.394
	C0321-multiple-statements	0.008	0.003	*C0209-consider-using-f-string*	0.024	0.024
	C0209-consider-using-f-string	0.005	0.003	C2401-non-ascii-name	0.014	0.014
	C0200-consider-using-enumerate	0.004	0.003	*C0200-consider-using-enumerate*	0.014	0.014
	C0325-superfluous-parens	0.003	0.003	*C0325-superfluous-parens*	0.005	0.005
Warning (W)	W0621-redefined-outer-name	0.165	0.070	W1203-logging-fstring-interpolation	0.221	0.047
	W0613-unused-argument	0.090	0.048	*W0613-unused-argument*	0.103	0.052
	W0612-unused-variable	0.064	0.042	W0718-broad-exception-caught	0.099	0.094
	W1514-unspecified-encoding	0.060	0.041	*W0612-unused-variable*	0.099	0.052
	W0718-broad-exception-caught	0.021	0.018	W1309-f-string-without-interpolation	0.094	0.056
Refactoring (R)	*R0903-too-few-public-methods*	0.088	0.063	*R0903-too-few-public-methods*	0.117	0.094
	R1705-no-else-return	0.033	0.031	*R1705-no-else-return*	0.042	0.033
	R0913-too-many-arguments	0.010	0.009	R1714-consider-using-in	0.038	0.028
	R1710-inconsistent-return-statements	0.009	0.009	R1735-use-dict-literal	0.019	0.019
	R1725-super-with-arguments	0.008	0.008	R1732-consider-using-with	0.014	0.014

frequent errors found in both ChatGPT-modified and ChatGPT-generated code. However, the γ_i and ν_i values show clear differences in their frequency of occurrences. For example, the modified code has nearly double the γ_i and higher ν_i for E0602 error. On the other hand, the E0001 in ChatGPT-generated code is nearly seven times more frequent than in ChatGPT-modified code. Similar scenarios are also observed for the other types of issues. Based on the observations, we now derive the answer to RQ1 as follows:

Ans. to RQ1: *ChatGPT-modified code is more prone to code quality issues than ChatGPT-generated code and developer-provided code. There is a possibility that ChatGPT's modifications to developer-provided code may introduce more quality issues. There are substantial overlaps among the most frequent issues found in ChatGPT-generated and ChatGPT-modified code.*

3.2 Assessment of Security Vulnerabilities

For both the ChatGPT-Generated and ChatGPT-modified code, in Table 3, we separately present the captured security issues and their corresponding CWE types along with the α_τ for CWEs and γ_i values for individual issues. The table is sorted to α_τ. As seen in the table, 19 distinct security issues related to 10 unique CWE types are found in ChatGPT-Generated, while only seven distinct security issues related to six unique CWEs are found in ChatGPT-modified code. All the six CWEs reflected in the ChatGPT-modified code are also reflected in the ChatGPT-Generated code. A more diverse set of security issues and CWEs are found in the ChatGPT-Generated code likely because this set contains 1,543 code snippets compared to only 213 ChatGPT-modified code snippets.

Interestingly, the security issue B113 under CWE-400 is found the most frequent in both ChatGPT-Generated and ChatGPT-modified code when we consider the α_τ and γ_i values. The order of the remaining CWEs differs between the two sets of code snippets. The top three CWEs reflected in ChatGPT-modified code (i.e., CWE-400, 703, 78) are found more frequently in ChatGPT-modified code compared to the ChatGPT-generated code as inferred from the values of α_τ. However, not much difference is observed for the rest three CWEs.

The higher frequency of the three CWEs in ChatGPT-modified code, may give a wrong impression that such code snippets are

Table 3: Security vulnerabilities in ChatGPT-aided code

Type (τ)	α_τ	Issues	γ_i
In ChatGPT-generated code			
CWE-400	0.021	B113:request_without_timeout	0.021
CWE-78	0.019	B404:blacklist (import_subprocess)	0.006
		B603:subprocess_without_shell_equals_true	0.004
		B607:start_process_with_partial_path	0.004
		B602:subprocess_popen_with_shell_equals_true	0.002
		B601:paramiko_calls	0.002
		B102:exec_used	0.001
CWE-259	0.016	B105:hardcoded_password_string	0.014
		B106:hardcoded_password_funcarg	0.002
CWE-703	0.016	B101:assert_used	0.016
CWE-94	0.013	B201:flask_debug_true	0.012
		B701:jinja2_autoescape_false	0.001
CWE-330	0.011	B311:blacklist (random)	0.011
CWE-327	0.007	B324:hashlib	0.005
		B413:blacklist (import_pycrypto)	0.002
CWE-502	0.006	B403:blacklist (import_pickle)	0.003
		B301:blacklist (pickle)	0.003
CWE-605	0.001	B104:hardcoded_bind_all_interfaces	0.001
CWE-20	0.001	B506:yaml_load	0.001
In ChatGPT-modified code			
CWE-400	0.052	B113:request_without_timeout	0.052
CWE-703	0.038	B101:assert_used	0.038
CWE-78	0.028	B404:blacklist (import_subprocess)	0.014
		B602:subprocess_popen_with_shell_equals_true	0.014
CWE-94	0.014	B201:flask_debug_true	0.014
CWE-330	0.009	B311:blacklist (random)	0.009
CWE-259	0.005	B105:hardcoded_password_string	0.005

more susceptible to security issues compared to ChatGPT-generated code. Hence, to further investigate, we capture the security vulnerabilities in the developer-provided code corresponding to the ChatGPT-modified counterparts. In Figure 2, we present the α_τ values computed for all six CWEs.

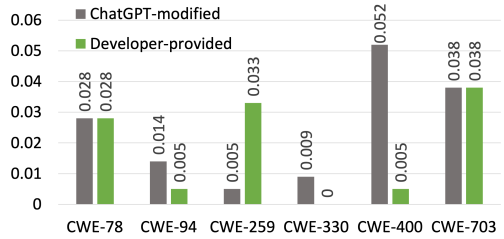

Figure 2: Security issues per code snippet (α_τ)

As seen in the figure, CWE-94, CWE-330, and CWE-400 are more frequently found in ChatGPT-modified code but CWE-259 is more frequently encountered in developer-provided code while CWE-78 and CWE-703 are equally present in both categories of code. Thus, here we cannot infer whether the security vulnerabilities were originally introduced by ChatGPT, which could have made the ChatGPT-modified code contain more security issues.

3.2.1 Relating Mitre's Top 25. The Mitre Corporation maintains a list [9] of "Top 25 Most Dangerous Software Weaknesses," which is a list of CWEs updated annually. Mitre also maintains another list of "stubborn weaknesses" [10] that includes those CWEs that consistently appear throughout the last five years' Top 25 Most Dangerous Software Weaknesses.

Table 4: Code snippets with CWEs in Mitre's 2023 top 25

Type (τ)	β_τ	Rank*
In ChatGPT-generated code		
CWE-78	0.009	5
CWE-20	0.001	6
CWE-502	0.003	15
CWE-94	0.013	23

Type (τ)	β_τ	Rank*
In ChatGPT-modified code		
CWE-78	0.014	5
CWE-94	0.014	23

*rank in the current list of top 25 [7] (lower rank implies more dangerous).

As shown in Table 4, out of the 10 unique CWEs identified in the ChatGPT-generated code, four of them are among Mitre's current (2023) top 25 most dangerous software weaknesses [7], whereas two of the six identified CWEs in ChatGPT-modified code are in the list. Among the 15 "stubborn weaknesses", three (i.e., CWE-78, 20, 502) are found in the ChatGPT-generated code, while one (CWE-78) found in ChatGPT-modified code. Again, the larger size of the set of the ChatGPT-generated snippets is a possible explanation for why this particular set shares more CWEs with the current top 25 most dangerous software weaknesses and 15 "stubborn weaknesses." The similar β_τ values further indicate no substantial differences of those CWEs' appearances in either set of code. Based on the above observations, we now derive the answer to RQ2 as follows:

Ans. to RQ2: *Concerning security issues, there are no significant differences between ChatGPT-generated and ChatGPT-modified code. The security vulnerabilities found in ChatGPT-modified code are sometimes introduced by ChatGPT while sometimes they previously existed in the developer-provided code.*

4 THREATS TO VALIDITY

One significant factor that can impact the validity of our analysis is the reliability of the tools, Pylint and Bandit, we employ. However, Pylint was reported to have 100% precision and Bandit was reported to have 90.79% precision [38]. Additionally, our analyses have not taken into account the sizes of the individual code snippets as we have seen that the sizes of the code snippets we encountered have not varied much. Our study focuses on Python code only. Thus, the conclusions drawn from this work may not apply to other programming languages. The findings are derived entirely based on quantitative analyses. Some qualitative insights could be useful in deepening our understanding of the results. Lastly, we restrict our focus to security vulnerabilities enumerated in the CWE list. Despite being a well-regarded catalog of weaknesses used widely in security research and industry, it may not represent an exhaustive compilation of every possible security issue.

5 RELATED WORK

There are many studies [2, 5, 6, 17–25, 36, 43, 43, 44] of code written by humans, but the studies of AI-generated code are scarce. AI code generation tools like Copilot [14] and Codex [32] have been found capable of generating *functionally accurate* code [11, 30, 42] and the presence of common bugs and security vulnerabilities [13, 33, 38]. Aljanabi et al. [1] highlighted ChatGPT's untapped potential for code generation, while Avila et al. [3] evaluated its skill at web-based tasks. Liu et al. [27] identified quality problems in ChatGPT's code output, from compilation errors to maintainability difficulties.

Nair et al. [29] studied ChatGPT's capabilities for hardware code generation and emphasized the need for careful prompting to avoid the generation of insecure code. Feng et al. [12] used crowdsourcing data to evaluate Python code from ChatGPT, identifying common errors. Khoury et al. [26] examined ChatGPT's understanding of security concerns, observing occasional non-robust code generation.

Our study stands out with a focus on comparative analysis of the quality and security vulnerabilities in ChatGPT-generated and ChatGPT-modified Python code including further comparisons with the original developer-provided code in the prompts.

6 CONCLUSION

In this paper, we have presented a quantitative study of the security and quality issues in Python code produced with assistance from ChatGPT. The code snippets are extracted from analyzing developers' conversations involving 17,622 prompts and ChatGPT responses in the DevGPT dataset [41].

Using a set of four metrics, we analyze the 1,756 Python code snippets categorized as either ChatGPT-generated code or ChatGPT-modified code produced from modifications to developer-provided original snippets. We find relatively more code quality issues in ChatGPT-modified code compared to ChatGPT-generated code. On the contrary, both categories of code almost equally include security vulnerabilities.

These findings imply that there is ample room for AI tools to improve in minimizing quality and security issues while generating and editing source code. The results also advocate for caution in using AI-aided code in software projects. In the future, we plan to extend this study by incorporating code written in diverse programming languages and also by deriving deeper insights through qualitative investigations.

ACKNOWLEDGEMENT

This work is supported in part by the ISU-CAES Seed Grant at the Idaho State University, USA.

REFERENCES

[1] Mohammad Aljanabi, Mohanad Ghazi, Ahmed Hussein Ali, Saad Abas Abed, et al. 2023. ChatGPT: open possibilities. *Iraqi Journal For Computer Science and Mathematics* 4, 1 (2023), 62–64.

[2] D. Alwad, M. Panta, and M. Zibran. 2018. An Empirical Study of the Relationships between Code Readability and Software Complexity. In *27th International Conference on Software Engineering and Data Engineering*. 122–127.

[3] Laurent Avila-Chauvet, Diana Mejía, and Christian Oswaldo Acosta Quiroz. 2023. Chatgpt as a support tool for online behavioral task programming. *Available at SSRN 4329020* (2023).

[4] Erik Brynjolfsson, Danielle Li, and Lindsey R Raymond. 2023. *Generative AI at Work*. Working Paper 31161. National Bureau of Economic Research. https://doi.org/10.3386/w31161

[5] A. Champa, M. Rabbi, M. Zibran, and M. Islam. 2023. Insights into Female Contributions in Open-Source Projects. In *20th IEEE International Conference on Mining Software Repositories*. 357–361.

[6] Arifa I. Champa, Md Fazle Rabbi, Costain Nachuma, and Minhaz F. Zibran. 2024. ChatGPT in Action: Analyzing Its Use in Software Development. In *Proceedings of the 21st International Conference on Mining Software Repositories (MSR 2024)*.

[7] The MITRE Corporation. 2023. *2023 CWE Top 25 Most Dangerous Software Weaknesses*. Retrieved December, 2023 from https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html

[8] The MITRE Corporation. 2023. *CWE - Common Weakness Enumeration*. Retrieved December 2023 from https://cwe.mitre.org/

[9] The MITRE Corporation. 2023. *CWE Top 25 Most Dangerous Software Weaknesses*. Retrieved December, 2023 from https://cwe.mitre.org/top25/

[10] The MITRE Corporation. 2023. *Stubborn Weaknesses in the CWE Top 25*. Retrieved December 2023 from https://cwe.mitre.org/top25/archive/2023/2023_stubborn_weaknesses.html

[11] Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated repair of programs from large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1469–1481.

[12] Yunhe Feng, Sreecharan Vanam, Manasa Cherukupally, Weijian Zheng, Meikang Qiu, and Haihua Chen. 2023. Investigating Code Generation Performance of ChatGPT with Crowdsourcing Social Data. In *Proceedings of the 47th IEEE Computer Software and Applications Conference*. 1–10.

[13] Yujia Fu, Peng Liang, Amjed Tahir, Zengyang Li, Mojtaba Shahin, and Jiaxin Yu. 2023. Security Weaknesses of Copilot Generated Code in GitHub. *arXiv preprint arXiv:2310.02059* (2023).

[14] GitHub. 2023. *GitHub copilot · your AI pair programmer*. Retrieved December 2023 from https://github.com/features/copilot

[15] GitHub. 2023. *The top programming languages*. Retrieved December 2023 from https://octoverse.github.com/2022/top-programming-languages

[16] Morey Haber. 2023. *Two Cybersecurity Concerns When Using ChatGPT For Software Development*. Retrieved December 2023 from https://www.forbes.com/sites/forbestechcouncil/2023/03/29/two-cybersecurity-concerns-when-using-chatgpt-for-software-development/

[17] M. Islam and M. Zibran. 2016. A Comparative Study on Vulnerabilities in Categories of Clones and Non-Cloned Code. In *10th IEEE Intl. Workshop on Software Clones*. 8–14.

[18] M. Islam and M. Zibran. 2016. Exploration and Exploitation of Developers' Sentimental Variations in Software Engineering. *Internation Journal of Software Innovation* 4, 4 (2016), 35–55.

[19] M. Islam and M. Zibran. 2016. Towards Understanding and Exploiting Developers' Emotional Variations in Software Engineering. In *SERA*. 185–192.

[20] M. Islam and M. Zibran. 2017. Leveraging Automated Sentiment Analysis in Software Engineering. In *MSR*. 203–214.

[21] M. Islam and M. Zibran. 2018. On the Characteristics of Buggy Code Clones: A Code Quality Perspective. In *12th IEEE Intl. Workshop on Software Clones*. 23 – 29.

[22] M. Islam and M. Zibran. 2018. Sentiment Analysis of Software Bug Related Commit Messages. In *27th Intl. Conference on Software Engineering and Data Engineering*. 3–8.

[23] M. Islam and M. Zibran. 2020. How Bugs Are Fixed: Exposing Bug-fix Patterns with Edits and Nesting Levels. In *35th ACM/SIGAPP Symposium on Applied Computing*. 1523–1531.

[24] M. Islam and M. Zibran. 2021. What Changes in Where? An Empirical Study of Bug-Fixing Change Patterns. *ACM Applied Computing Review* 20, 4 (2021), 18–34.

[25] M. Islam, M. Zibran, and A. Nagpal. 2017. Security Vulnerabilities in Categories of Clones and Non-Cloned Code: An Empirical Study. In *11th ACM/IEEE Intl. Symposium on Empirical Software Engineering and Measurement*. 20–29.

[26] Raphaël Khoury, Anderson R Avila, Jacob Brunelle, and Baba Mamadou Camara. 2023. How Secure is Code Generated by ChatGPT? *arXiv preprint arXiv:2304.09655* (2023).

[27] Yue Liu, Thanh Le-Cong, Ratnadira Widyasari, Chakkrit Tantithamthavorn, Li Li, Xuan-Bach D Le, and David Lo. 2023. Refining ChatGPT-Generated Code: Characterizing and Mitigating Code Quality Issues. *arXiv preprint arXiv:2307.12596* (2023).

[28] Christopher D Manning. 2009. *An introduction to information retrieval*. Cambridge university press.

[29] Madhav Nair, Rajat Sadhukhan, and Debdeep Mukhopadhyay. 2023. Generating secure hardware using chatgpt resistant to cwes. *Cryptology ePrint Archive* (2023).

[30] Nhan Nguyen and Sarah Nadi. 2022. An empirical evaluation of GitHub copilot's code suggestions. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 1–5.

[31] OpenAI. 2023. *Models Overview*. Retrieved December 2023 from https://platform.openai.com/docs/models/gpt-3-5/

[32] OpenAI. 2023. *OpenAI Codex*. Retrieved December 2023 from https://openai.com/blog/openai-codex

[33] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2022. Asleep at the keyboard? assessing the security of github copilot's code contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 754–768.

[34] PyPI. 2023. *Bandit 1.7.5 Project description*. Retrieved December 2023 from https://pypi.org/project/bandit/

[35] PyPI. 2023. *Pylint 3.0.2 Project description*. Retrieved December 2023 from https://pypi.org/project/pylint/

[36] Md Fazle Rabbi, Arifa I. Champa, Costain Nachuma, and Minhaz F. Zibran. 2024. SBOM Generation Tools Under Microscope: A Focus on the npm Ecosystem. In *In Proceedings of ACM Symposium on Applied Computing (SAC 2024)*.

[37] Inbal Shani and GitHub Staff. 2023. *Survey reveals AI's impact on the developer experience*. Retrieved December 2023 from https://github.blog/2023-06-13-survey-reveals-ais-impact-on-the-developer-experience/

[38] Mohammed Latif Siddiq, Shafayat H Majumder, Maisha R Mim, Sourov Jajodia, and Joanna CS Santos. 2022. An Empirical Study of Code Smells in Transformer-based Code Generation Techniques. In *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 71–82.

[39] Bart Van Oort, Luís Cruz, Maurício Aniche, and Arie Van Deursen. 2021. The prevalence of code smells in machine learning projects. In *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)*. IEEE, 1–8.

[40] Menghan Wu, Yang Zhang, Jiakun Liu, Shangwen Wang, Zhang Zhang, Xin Xia, and Xinjun Mao. 2022. On the way to microservices: Exploring problems and solutions from online q&a community. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 432–443.

[41] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. 2024. DevGPT: Studying Developer-ChatGPT Conversations. In *Proceedings of the International Conference on Mining Software Repositories (MSR 2024)*.

[42] Burak Yetistiren, Isik Ozsoy, and Eray Tuzun. 2022. Assessing the quality of GitHub copilot's code generation. In *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*. 62–71.

[43] M. Zibran and C. Roy. 2013. Conflict-aware Optimal Scheduling of Code Clone Refactoring. *IET Software* 7, 3 (2013), 167–186.

[44] M. Zibran, R. Saha, C. Roy, and K. Schneider. 2013. Evaluating the Conventional Wisdom in Clone Removal: A Genealogy-based Empirical Study. In *28th ACM/SIGAPP Symposium on Applied Computing*. 1123–1130.