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:

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.

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-modified. Thus, there remains a substantial similarity between a developer-provided code snippet and the corresponding ChatGPT-modified code snippet.
Hence, to distinguish the ChatGPT-modified 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-modified 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-modified 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-modified 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 $\tau$) 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 $\tau$, the average number of issues of type $\tau$ per code snippet, denoted by $\alpha_\tau$, is calculated as:
  \[ \alpha_\tau = \frac{\text{Total occurrences of issues of type } \tau}{\text{Total number of code snippets}} \]

- For each issue type $\tau$, the proportion of code snippets with at least one issue of type $\tau$, denoted as $\beta_\tau$, is obtained by:
  \[ \beta_\tau = \frac{\text{Total code snippets with issues of type } \tau}{\text{Total number of code snippets}} \]

- For each issue $i$, the average number of occurrences of issue $i$ across snippets, denoted as $y_i$, is computed as:
  \[ y_i = \frac{\text{Number of occurrences of } i}{\text{Total number of code snippets}} \]

- For each issue $i$, the proportion of code snippets having one or more occurrences of issue $i$, denoted as $v_i$, is defined as:
  \[ v_i = \frac{\text{Total code snippets having the issue } i}{\text{Total number of code snippets}} \]

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 $\alpha_\tau$ and $\beta_\tau$ 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

<table>
<thead>
<tr>
<th>Type</th>
<th>In ChatGPT-generated Code</th>
<th>In ChatGPT-modified Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>$\alpha_E = 1.70$</td>
<td>$\alpha_E = 2.68$</td>
</tr>
<tr>
<td>C</td>
<td>$\beta_C = 0.27$</td>
<td>$\beta_C = 0.31$</td>
</tr>
<tr>
<td>W</td>
<td>$\alpha_W = 0.53$</td>
<td>$\alpha_W = 0.68$</td>
</tr>
<tr>
<td>R</td>
<td>$\beta_R = 0.18$</td>
<td>$\beta_R = 0.26$</td>
</tr>
</tbody>
</table>

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 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 $y_i$ and $v_i$ metric values for each of the 69 distinct 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 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

<table>
<thead>
<tr>
<th>Type (r)</th>
<th>Issue</th>
<th>In ChatGPT-generated Code</th>
<th>In ChatGPT-modified Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error (E)</td>
<td>E0602-undefined-variable</td>
<td>1.475 0.409</td>
<td>2.939 0.502</td>
</tr>
<tr>
<td></td>
<td>E0001-syNTAX-error</td>
<td>0.099 0.099</td>
<td>0.080 0.038</td>
</tr>
<tr>
<td></td>
<td>E1101-no-member</td>
<td>0.084 0.029</td>
<td>0.019 0.019</td>
</tr>
<tr>
<td></td>
<td>E0611-no-name-in-module</td>
<td>0.014 0.011</td>
<td>0.014 0.014</td>
</tr>
<tr>
<td></td>
<td>R0104-return-outside-function</td>
<td>0.008 0.007</td>
<td>0.009 0.009</td>
</tr>
<tr>
<td>Convention (C)</td>
<td>C0301-line-too-long</td>
<td>0.251 0.126</td>
<td>1.249 0.394</td>
</tr>
<tr>
<td></td>
<td>C0321-multiple-statements</td>
<td>0.008 0.003</td>
<td>0.024 0.024</td>
</tr>
<tr>
<td></td>
<td>C0209-consider-using-f-string</td>
<td>0.005 0.005</td>
<td>0.024 0.024</td>
</tr>
<tr>
<td></td>
<td>C0200-consider-using-enumerate</td>
<td>0.004 0.005</td>
<td>0.024 0.024</td>
</tr>
<tr>
<td></td>
<td>C0325-superfluous-parens</td>
<td>0.003 0.005</td>
<td>0.024 0.024</td>
</tr>
<tr>
<td>Warning (W)</td>
<td>W0621-redirected-outer-name</td>
<td>0.163 0.070</td>
<td>0.221 0.047</td>
</tr>
<tr>
<td></td>
<td>W0613-unused-argument</td>
<td>0.090 0.048</td>
<td>0.103 0.052</td>
</tr>
<tr>
<td></td>
<td>W0612-unused-variable</td>
<td>0.064 0.042</td>
<td>0.099 0.094</td>
</tr>
<tr>
<td></td>
<td>W1514-unsupported-encoding</td>
<td>0.060 0.041</td>
<td>0.099 0.094</td>
</tr>
<tr>
<td></td>
<td>W0718-broad-exception-caught</td>
<td>0.021 0.018</td>
<td>0.094 0.056</td>
</tr>
<tr>
<td>Refactoring (R)</td>
<td>R0903-too-few-public-methods</td>
<td>0.088 0.063</td>
<td>0.117 0.094</td>
</tr>
<tr>
<td></td>
<td>R1705-no-else-return</td>
<td>0.033 0.031</td>
<td>0.042 0.035</td>
</tr>
<tr>
<td></td>
<td>R0913-too-many-arguments</td>
<td>0.010 0.009</td>
<td>0.038 0.028</td>
</tr>
<tr>
<td></td>
<td>R1710-inconsistent-return-statements</td>
<td>0.009 0.009</td>
<td>0.019 0.019</td>
</tr>
<tr>
<td></td>
<td>R1725-super-with-arguments</td>
<td>0.008 0.008</td>
<td>0.014 0.014</td>
</tr>
</tbody>
</table>

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 $\alpha_i$ for CWEs and $\gamma_i$ values for individual issues. The table is sorted to $\alpha_i$. As seen in the table, 19 distinct security issues related to 10 unique CWE types are found in ChatGPT-Generated code, 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 in both ChatGPT-Generated and ChatGPT-modified code when we consider the $\alpha_i$ and $\gamma_i$ 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 $\alpha_i$. 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...
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 $\alpha$ values computed for all six CWEs.

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 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.

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 $\beta$ 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, 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