A Quantitative Analysis of Quality and Consistency in AI-generated Code

Autumn Clark Daniel Igbokwe Samantha Ross Minhaz F. Zibran
Department of Computer Science, Idaho State University
Pocatello, ID, USA
{autumnclark, danieligbokwe, samanthaross, minhazzibran}@isu.edu

***Abstract*—With the recent emergence of generative AI (Artificial intelligence), Large Language Model (LLM) based tools such as ChatGPT have become popular assistants to humans in diverse tasks. ChatGPT has also been widely adopted for solving programming problems and for generating source code in software development. This research investigates both the code quality and the consistency of code quality over iterative prompts in 625 ChatGPT-generated Python code samples in the DevGPT dataset and the corresponding code snippets regenerated by manually prompting ChatGPT. Code samples are measured in terms of seven Halstead complexity metrics. We also assess how consistent they are across code snippets generated by different versions of ChatGPT. It was found that while ChatGPT generates good quality code across iterative prompts, it does generate semi-frequent bugs, similar to how humans do, necessitating code review before integration. These traits also remain consistent across code snippets generated by subsequent releases of ChatGPT. These results suggest using AI-generated source code in software development will not hinder the process.**

***Index Terms*—ChatGPT, Generative AI, Code, Quality, Complexity, Consistency, Python, Program, Analysis**

I. Introduction

The rapid development of large language models (LLMs) has elevated natural language processing, innovating new generative artificial intelligence (AI) that is increasingly creative and convenient. Among these, ChatGPT has emerged as a powerful tool for generating responses to incredibly diverse prompts. As ChatGPT offers a new approach to code generation, understanding the inherent quality of the code it produces becomes paramount for developers and researchers alike.

Furthermore, the consistent generation of reliable code is a critical aspect to address, especially considering the growing integration of AI in software development processes. However, assessing the quality and consistency of code generated by LLMs like ChatGPT is not a smooth undertaking. Due to the dynamic nature of programming, it is difficult to evaluate code quality generated by ChatGPT. Code quality cannot be assessed solely on correctness; it requires an understanding of the structural complexity and adherence to established standards. Similarly, code consistency from generative models, such as ChatGPT, is also difficult to evaluate since responses can provide the same functionality in various approaches.

As these AI tools are new, very little is known about the quality and consistency of the source code generated by such tools. However, this understanding is necessary for informing AI researchers with directions to further enhance their tools and also to inform the programmers and software practitioners with a deeper understanding of how to effectively use the AI-generated source code in their projects. Therefore, in this work, we study the quality and consistency of AI-generated source code. In particular, we address the following two research questions.

RQ1: What quality and complexity level of source code does ChatGPT generate?

RQ2: How consistent is source code generated by different versions of ChatGPT in addressing identical issues?

The answer to RQ1 will allow for a better understanding of the overall quality of code generated by ChatGPT. This information will allow developers to make informed decisions about the scenarios in which using generated code is appropriate and how much that code needs to be analyzed before use. The answer to RQ2 will allow developers to know if the same prompt will always produce the same code, or if repetitive prompting will be necessary to achieve previous results.

To address the aforementioned research questions, we evaluate the quality and consistency of ChatGPT-generated code extracted from developer-ChatGPT conversations in the DevGPT dataset [1]. Among the alternative AI tools, we pick ChatGPT because this generative AI tool has recently come to the limelight with its diverse abilities including code generation. Moreover, there is a recently released public dataset, DevGPT [1], consisting of a collection of conversations between developers and ChatGPT, which serves the purpose of our work.

The rest of the paper is organized as follows. The methodology of our study is described in Section II. The results and analysis are discussed in Section III. The threats to validity are discussed in Section IV. Prior research in the literature related to our work is discussed in Section V. Finally, Section VI concludes the paper with future research directions in the area.

II. Methodology

The procedural steps involved in data collection and analyses for our study are summarized in Figure 1.

A. Dataset

The DevGPT [1] dataset used in this work is a public repository on GitHub. This dataset is structured in 54 JSON

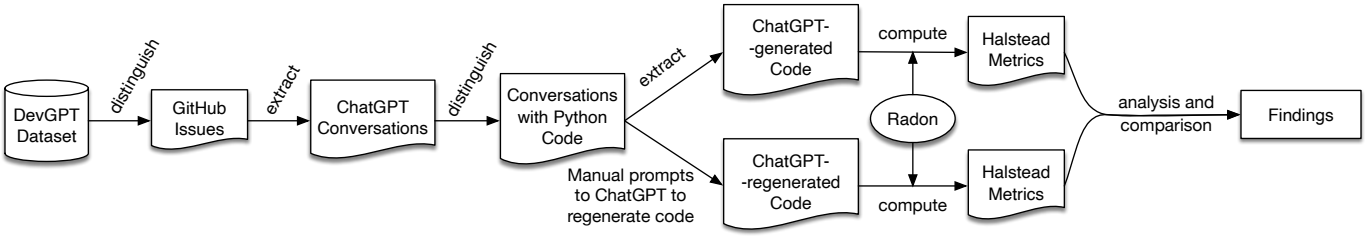

Fig. 1. Procedural Steps in the Methodology of Our Study

files over nine snapshots of developer-ChatGPT conversations. These conversations are organized into six categories as they are drawn from the corresponding six sources: 'GitHub Issues', 'Pull Requests (PR)', 'Discussions', 'Commits', 'Code Files', and 'Hacker News Threads'. Table I summarizes the DevGPT dataset content, particularly showing the number of conversations with any code, the total number of code snippets, and the total number of prompts included in each of the six categories/sources of conversations.

TABLE I
AN OVERVIEW OF THE DEVGPT [1] DATASET

Sources / Categories	# of Conversations with Code	# of Code Snippets	# of Prompts
Hacker News Threads	66	309	1,434
GitHub Issues	317	1,863	2,077
Pull Requests (PR)	559	1,948	1,636
Commits	153	1,022	1,041
Code Files	538	6,669	11,214
Discussions	34	220	220
Total	1,667	12,031	17,622

As shown in Figure 2, every source within the dataset includes a 'ChatGPTSharing' attribute, which contains a list of 'Conversations'. Each conversation contains the following attributes:

Prompt represents the questions/requests made by the developers to ChatGPT.

Answer includes ChatGPT's responses to developers' questions in the prompts.

List of Code includes the list of code snippets as parts of the answers in CHatGPT's responses.

The DevGPT data was gathered every week in the year 2023 starting July 27th and ending August 31st. To distinguish a manageable portion of the DevGPT dataset, our study focuses on the ChatGPT conversations of the category 'GitHub Issues' in nine JSON files across all nine DevGPT snapshots. The JSON file's nested structure is detailed in Figure 2.

B. Metrics

To measure the complexity and quality of the Python code snippets, we compute the Halstead complexity metrics. The Halstead Complexity Measures provide a robust framework offering a nuanced perspective capable of quantifying the quality of generated code. Invented by Maurice Howard Halstead in 1977 [2], the metrics are based on defining the mathematical relationships between the distinct operators and operands of a program. It is one of the most popular metrics for measuring code density [3]. The Halstead Measures can also be used to assess consistency by comparing the metric scores across responses to the same query.

The Halstead Complexity Measure includes eight metrics to estimate the quality of a given code snippet. These eight metrics are volume, difficulty, length, calculated length, effort, time required to program, number of delivered bugs, and vocabulary. Vocabulary was excluded from this study as it merely provides the number of terms used as the sum of distinct operators and operands, which was not useful information for this research. The calculations for the seven metrics in the Halstead complexity measure used in our study are presented in Table II.

TABLE II
HALSTEAD COMPLEXITY METRICS USED IN OUR STUDY

Metric	Notation	Computation
Program Length	$\mathcal{N}$	$\mathcal{N} = N_1 + N_2$
Calculated Program Length	$\mathcal{L}$	$\mathcal{L} = n_1 log_2 n_1 + n_2 log_2 n_2$
Volume	$\mathcal{V}$	$\mathcal{V} = \mathcal{N} log_2(n_1 + n_2)$
Difficulty	$\mathcal{D}$	$\mathcal{D} = \frac{n_1}{2} \times \frac{N_2}{n_2}$
Effort	$\mathcal{E}$	$\mathcal{E} = \mathcal{D} \times \mathcal{V}$
Time Required To Program	$\mathcal{T}$	$\mathcal{T} = \frac{\mathcal{E}}{18} seconds$
Number of Delivered Bugs	$\mathcal{B}$	$\mathcal{B} = \frac{\mathcal{V}}{3000}$

Here, n_1 = number of distinct operators
n_2 = number of distinct operands
N_1 = total number of operators
N_2 = total number of operands

For computing the Halstead complexity metrics, we use Radon [4], which is a Python tool that evaluates source code using a variety of metrics, including the Halstead evaluation. The tool is utilized via the command line and it is compatible with Python versions 2.7 to 3.8. The use of this third-party software tool for computing the metrics helps avoid human subjectivity and maintain consistency.

III. ANALYSIS AND FINDINGS

The code snippets in the DevGPT dataset are found written in 109 different languages. We confine our analysis to code

TABLE III
HALSTEAD METRICS COMPUTED FOR GENERATED, REGENERATED PYTHON CODE, AND THEIR DIFFERENCES

Halstead Metrics	Average Metric Values Computed for Generated Code	for Regenerated Code	Difference ($generated - regenerated$)
Program Length ($\mathcal{N}$)	11.49	11.58	-0.09
Calculated Program Length ($\mathcal{L}$)	28.35	27.85	0.50
Volume ($\mathcal{V}$)	54.28	50.93	3.35
Difficulty ($\mathcal{D}$)	19.45	18.48	0.97
Effort ($\mathcal{E}$)	158.84	106.40	52.44
Time Required to Program ($\mathcal{T}$)	61.41	41.03	20.38
Number of Delivered Bugs ($\mathcal{B}$)	20.49	13.70	6.79

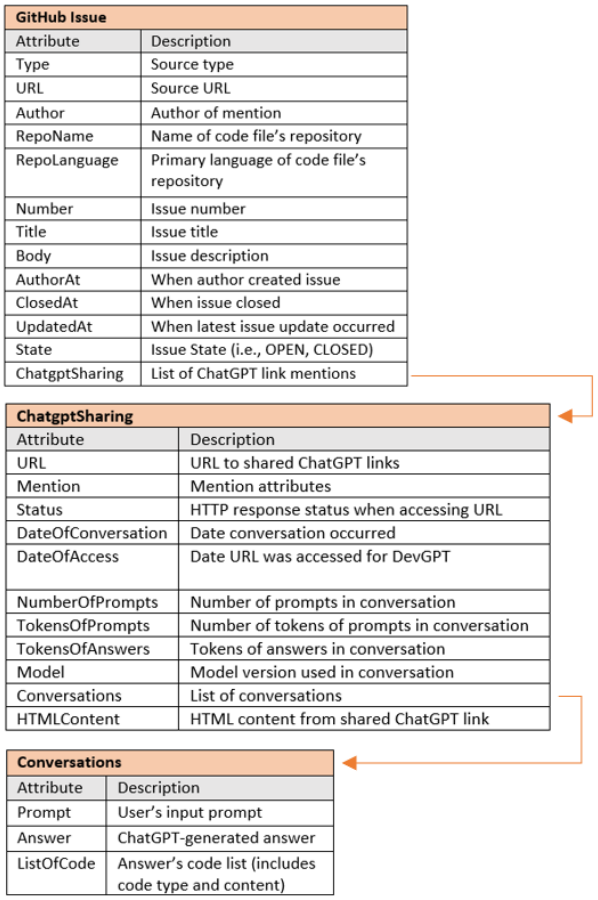

Fig. 2. The Structure of an Issue Sharing File in the DevGPT Dataset

written in Python only. 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 [5]. Second, Python code is also dominant in the chosen snapshot of the DevGPT dataset; 1,756 of the total 12,031 code snippets are found written in Python.

A. Extraction and Assessment of Python Code

We filter the DevGPT dataset down to only data points where the repository language was identified as Python. From there, conversations are iterated over and individual code blocks are extracted and saved as individual Python files. Then Radon is operated on each of these files for computing the seven Halstead metrics. Files that were incompatible with Radon were removed from the project's dataset. Thus, a toal of 625 Python code snippets remain in our study for further analysis. The average values of the seven Halstead metrics for these 625 ChatGPT-generated Python code snippets are presented in the second column from the left in Table III.

Answer to RQ1: The code provided by ChatGPT, on average, is relatively short in length, with an average program length ($\mathcal{N}$) of 11.49. However, the effort required to write the code and the time required to do so is relatively high, with an average effort ($\mathcal{E}$) of 158.84 and an average time required to program ($\mathcal{T}$) of 61.41. This shows that the code generated by ChatGPT is likely dense and complex, indicating that ChatGPT can provide far more than rudimentary code. It is worth noting that, on average, the number of delivered bugs ($\mathcal{B}$) is 20.49, indicating that ChatGPT makes semi-frequent mistakes.

B. Measuring ChatGPT's Consistency

Note that all of the 625 ChatGPT-generated Python code snippets described before are extracted from the DevGPT dataset, which includes developers' conversations with ChatGPT version 3.4. At the time of performing this study, ChatGPT is updated to version 3.5. To address RQ2, we want to understand how consistent the source code generated by subsequent versions of ChatGPT is in dealing with the same request/question from a developer.

Hence, for each of the 625 ChatGPT-generated Python code snippets, the URL links to the ChatGPT discussions are accumulated. The original conversations were replicated by manually feeding the prompts into ChatGPT for each conversation. The code generated by ChatGPT 3.5 as parts of the answers/responses are recorded, which we refer to as 'regenerated' code. All the seven Halstead metrics are then computed using Radon for each of the regenerated code snippets. The values for these seven metrics separately averaged over all the regenerated code snippets are presented in the second column from the right in Table III. The difference between the Halstead metrics for the ChatGPT-generated code snippets and the ChatGPT-regenerated code snippets are shown

in the rightmost column in Table III. For convenience in comparisons, in Figure 3, these metric values for the ChatGPT-generated and ChatGPT-regenerated are also plotted in a column graph side by side.

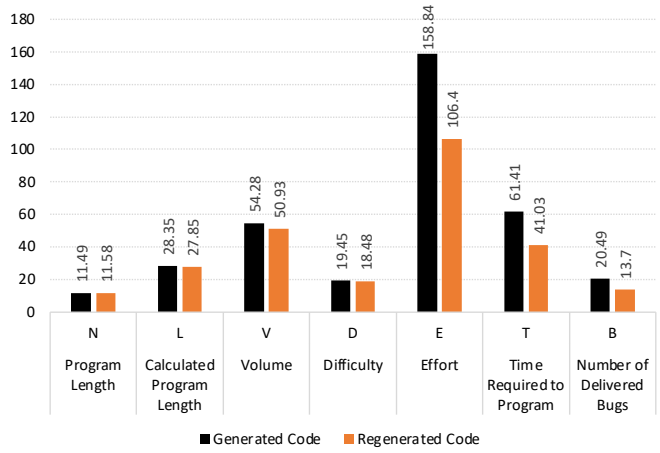

Fig. 3. Halstead Metrics Computed for Generated and Regenrated Code

As observed in Table III and in Figure 3, for all the seven metrics, the values are lower for ChatGPT-regenerated code compared to those for the ChatGPT-generated code. The only exception is for program length ($\mathcal{N}$) where the average program length ($\mathcal{N}$) for ChatGPT-regenerated code is slightly higher, by 0.09 only. This implies that ChatGPT has improved in code generation from version 3.4 to 3.5.

The differences between the ChatGPT-generated code and ChatGPT-regenerated code in the average program length ($\mathcal{N}$), calculated program length ($\mathcal{L}$), and difficulty ($\mathcal{D}$) are all near zero. This indicates that ChatGPT is very consistent with the length and difficulty of its generated code. The differences in average volume ($\mathcal{V}$) and the number of delivered bugs ($\mathcal{B}$) are not as close to zero, but still quite small, indicating that ChatGPT is somewhat consistent about the volume and number of bugs of its generated code. However, the differences in effort ($\mathcal{E}$) and the time required to program ($\mathcal{T}$) are relatively large, especially for $\mathcal{E}$, indicating that across the subsequent versions of ChatGPT, its code generation capabilities have substantially improved in these two criteria.

Answer to RQ2: The observations discussed above indicate that ChatGPT is very inconsistent about the effort required to reproduce its code and rather inconsistent about the time required to reproduce its code. When analyzing all of the average metrics, the overall quality of code provided by ChatGPT from iterative prompts appears to be very consistent.

IV. Threats To Validity

This work has used Radon as a means of calculating the Halstead metrics for the code snippets, of which only compatible files could be calculated. This has resulted in some code snippets being excluded from our study. This subsequently has affected the viability of samples from the regenerated code obtained from iteratively prompting ChatGPT. As such, careful consideration and analysis of the exact impact this had is warranted.

Halstead complexity metrics may not have the capacity to capture all aspects of complexity and quality of source code. To keep the dataset manageable, our study deals with only the conversations in the 'GitHub Issue' category/source. These can be argued as limitations of our work.

In producing the ChatGPT-regenerated code snippets, the authors have manually entered the corresponding original prompts into ChatGPT. Manual prompt entry is susceptible to human error, and it may have affected the results of this study.

Our study deals with Python code generated by ChatGPT only. Thus, the findings may not be generalizable to similar aspects of ChatGPT in the generation of source code in other programming languages or to other AI tools similar to ChatGPT.

The DevGPT dataset, Radon, and ChatGPT used in this work are publicly available. The methodology for data collection and analysis is documented in this paper in detail. Thus, it should be possible to replicate this study.

V. Related Work

The field of code generation with LLMs has witnessed substantial growth, with models like ChatGPT demonstrating remarkable capabilities in generating code snippets based on natural language prompts. Previous research has explored the potential of LLMs in generating diverse and complex code, setting the stage for our investigation into the consistency and quality of code produced by ChatGPT.

Evaluating the output of LLMs in code generation tasks is critical to understanding their practical utility. Existing approaches largely evaluate code based on functional correctness. EvalPlus, a recently created Python tool that evaluates LLM-synthesized code, enhances the evaluation process by enriching standard benchmark code test cases through LLMs and mutations [6]. The Security Assessment of LLMs (SALLM) research develops a framework to assess security concerns of LLM-generated code [7]. Our work builds upon these evaluation methods by focusing on the complexity of code generated by ChatGPT. We employ Halstead metrics to assess code complexity comprehensively, including parameters such as program length, volume, and difficulty.

The consistency of responses from LLMs to developer prompts is an essential aspect of understanding the reliability and robustness of these models in a developer-centric context. While previous studies have touched on aspects of semantic equivalence, negation, symmetric, and transitive consistency [8], our work takes a focused quantitative approach to analyze the complexity of ChatGPT's responses to the same prompts.

There are many studies [9], [10], [11], [12], [13], [14], [9], [15], [16], [17], [18], [19], [20], [21], [22] of code written by humans, but the studies of AI-generated code are scarce. Recently Champa et al. [23] analyzed the tasks developers seek assistance from ChatGPT and how effectively ChatGPT addresses them. Rabbi et al. [24] performed a comparative

analysis of ChatGPT-generated and ChatGPT-modified code. In contrast, our study investigates the quality of ChatGPT-generated code while we also examine the consistency of ChatGPT in addressing the same issue when fed to different versions of the AI tool.

VI. Conclusion

This study has looked into the quality of code generated by ChatGPT and the consistency of the quality of code generated by ChatGPT over iterative developer prompts from the DevGPT dataset. The results are derived from a quantitative analysis of Python code generated by ChatGPT and the corresponding regenerated code snippets obtained from manually prompting ChatGPT. The ChatGPT-generated code snippets are extracted from developer-ChatGPT conversations recorded in the publicly available DevGPT [1] dataset. The quality and complexity of the code snippets are measured in terms of seven Halstead complexity metrics using a state-of-the-art code analysis tool Radon [4]. This research shows that the code generated is fairly short in length, rather complex, and contains a moderate amount of bugs. This is indicative of a relatively high quality of code generated by ChatGPT, albeit with bugs present semi-frequently. This research also shows that the quality of code generated by ChatGPT is mostly consistent across iterative prompts.

From this, it can be concluded that ChatGPT, on average, generates code of good quality and tends to be consistent with the quality of the code it generates. Developers can use ChatGPT to assist in generating code and be confident that the code is of good quality and that the quality is consistent across uses. This means that using ChatGPT, and likely other LLMs, in the software development lifecycle will likely not hinder the process. However, ChatGPT was found to have generated code with bugs, similar to how humans do, so locating bugs and correcting them is still required.

A. Future Work

This research has focused on ChatGPT-generated code in a specific programming language with a specific evaluation metric. Future work in this domain could explore the quality and consistency of ChatGPT-generated code across multiple metrics and multiple programming languages. Cross-language analysis with multiple metrics would provide valuable insights into ChatGPT's performance and adaptability in various coding environments. The practicality of the generated code, as well as how much alteration is required before it can be incorporated into developer projects, would also be a promising avenue to investigate. Additionally, assessing the quality of generated code from ChatGPT compared to other sources, such as similar LLMs, would provide further insights into the strengths and weaknesses of generative models.

Acknowledgement

This work is supported in part by the ISU-CAES (Center for Advanced Energy Studies) Seed Grant at the Idaho State University (ISU), USA.

References

[1] T. Xiao, C. Treude, H. Hata, and K. Matsumoto, "Devgpt: Studying developer-chatgpt conversations," in *Proceedings of the International Conference on Mining Software Repositories (MSR 2024)*, 2024.

[2] *Software Science Revisited: Rationalizing Halstead's System Using Dimensionless Units*. US Department of Commerce, National Institute of Standards and Technology, 2018.

[3] J. Moreno-Leon, G. Robles, and M. Roman-Gonzalez, "Comparing computational thinking development assessment scores with software complexity metrics." in *IEEE global engineering education conference (EDUCON)*. IEEE, 2016, pp. 1040–1045.

[4] M. Lacchia, *Radon*. https://pypi.org/project/radon/ (Verified: Jan 2024).

[5] GitHub. (2023) The top programming languages. [Online]. Available: https://octoverse.github.com/2022/top-programming-languages

[6] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, *Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation*. https://arxiv.org/abs/2305.01210 (Verified: Jan 2024).

[7] M. L. Siddiq and J. C. Santos, *Generate and Pray: Using SALLMS to Evaluate the Security of LLM Generated Code*. https://arxiv.org/abs/2311.00889 (Verified: January 2024).

[8] M. E. Jang and T. Lukasiewicz, *Consistency Analysis of ChatGPT*. https://arxiv.org/abs/2303.06273 (Verified: January 2024).

[9] M. Zibran and C. Roy, "Conflict-aware optimal scheduling of code clone refactoring," *IET Software*, vol. 7, no. 3, pp. 167–186, 2013.

[10] M. Islam and M. Zibran, "How bugs are fixed: Exposing bug-fix patterns with edits and nesting levels," in *35th ACM/SIGAPP Symposium on Applied Computing*, 2020, pp. 1523–1531.

[11] M. Islam and M. Zibran, "What changes in where? an empirical study of bug-fixing change patterns," *ACM Applied Computing Review*, vol. 20, no. 4, pp. 18–34, 2021.

[12] M. Islam and M. Zibran, "A comparative study on vulnerabilities in categories of clones and non-cloned code," in *10th IEEE Intl. Workshop on Software Clones*, 2016, pp. 8–14.

[13] M. Islam, M. Zibran, and A. Nagpal, "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*, 2017, pp. 20–29.

[14] M. Zibran, R. Saha, C. Roy, and K. Schneider, "Evaluating the conventional wisdom in clone removal: A genealogy-based empirical study," in *28th ACM/SIGAPP Symposium on Applied Computing*, 2013, pp. 1123–1130.

[15] M. Islam and M. Zibran, "On the characteristics of buggy code clones: A code quality perspective," in *12th IEEE Intl. Workshop on Software Clones*, 2018, pp. 23 – 29.

[16] D. Alwad, M. Panta, and M. Zibran, "An empirical study of the relationships between code readability and software complexity," in *27th International Conference on Software Engineering and Data Engineering*, 2018, pp. 122–127.

[17] M. Islam and M. Zibran, "Towards understanding and exploiting developers' emotional variations in software engineering," in *SERA*, 2016, pp. 185–192.

[18] M. Islam and M. Zibran, "Exploration and exploitation of developers' sentimental variations in software engineering," *Internation Journal of Software Innovation*, vol. 4, no. 4, pp. 35–55, 2016.

[19] M. Islam and M. Zibran, "Sentiment analysis of software bug related commit messages," in *27th Intl. Conference on Software Engineering and Data Engineering*, 2018, pp. 3–8.

[20] M. Islam and M. Zibran, "Leveraging automated sentiment analysis in software engineering," in *MSR*, 2017, pp. 203–214.

[21] A. Champa, M. Rabbi, M. Zibran, and M. Islam, "Insights into female contributions in open-source projects," in *20th IEEE International Conference on Mining Software Repositories*, 2023, pp. 357–361.

[22] M. F. Rabbi, A. I. Champa, C. Nachuma, and M. F. Zibran, "SBOM generation tools under microscope: A focus on the npm ecosystem," in *In Proceedings of ACM Symposium on Applied Computing (SAC 2024)*, 2024.

[23] A. I. Champa, M. F. Rabbi, C. Nachuma, and M. F. Zibran, "ChatGPT in action: Analyzing its use in software development," in *Proceedings of the 21st International Conference on Mining Software Repositories (MSR 2024)*, 2024.

[24] M. F. Rabbi, A. I. Champa, M. F. Zibran, and M. R. Islam, "AI writes, we analyze: The ChatGPT python code saga," in *Proceedings of ACM International Conference on Mining Software Repositories (MSR 2024)*, 2024.