Can ChatGPT Support Developers? An Empirical Evaluation of Large Language Models for Code Generation

Kailun Jin, Chung-Yu Wang, Hung Viet Pham, Hadi Hemmati

{ktaming,cywang14,hvpham,hemmati}@yorku.ca

York University

Toronto, ON Canada

ABSTRACT

Large language models (LLMs) have demonstrated notable proficiency in code generation, with numerous prior studies showing their promising capabilities in various development scenarios. However, these studies mainly provide evaluations in research settings, which leaves a significant gap in understanding how effectively LLMs can support developers in real-world. To address this, we conducted an empirical analysis of conversations in DevGPT, a dataset collected from developers' conversations with ChatGPT (captured with the Share Link feature on platforms such as GitHub). Our empirical findings indicate that the current practice of using LLM-generated code is typically limited to either demonstrating high-level concepts or providing examples in documentation, rather than to be used as production-ready code. These findings indicate that there is much future work needed to improve LLMs in code generation before they can be integral parts of modern software development.

1 INTRODUCTION

The field of Artificial Intelligence (AI) has seen a major paradigm shift, characterized by powerful Large Language Models (LLMs) [6]. Recently, LLMs such as CodeGPT [13], CodeParrot [20], and Codex [4] have demonstrated their ability to facilitate code completion [25], source code mapping [10], system maintenance, [22] and other related Software Engineering tasks. Their contribution to software development is further reinforced by the iterative improvement through the collaboration between humans and AI [18].

However, amidst the promising developments, it is not clear how practical the integration of these LLMs into real-world production software development is, as prior work only demonstrates LLMs' potential in research settings. Specifically, the use of ChatGPT has not been studied as well as other code models yet [3]. Hence, it is imperative to critically evaluate the limitations and practicality of applying ChatGPT in real-world data practice.

To address this evaluation gap, this paper focuses on two key research questions (RQs):

- RQ1: How do the developers interact with ChatGPT for code generation?
- RQ2: How helpful is the code generated by ChatGPT in assisting developers?

RQ1 seeks to understand the structure of conversations between developers and ChatGPT during code generation which indicates the dynamics of collaboration and can affect the quality of code produced. It will also help identify future direction on how to better interact with LLMs through prompts, in this domain. RQ2 explores deeper into the usage of the generated code and studies to what extent the output of ChatGPT is useful for developers.

The main contributions of this work are:

- A detailed analysis of the real-world interactions between developers and ChatGPT
- An empirical examination of the usage of generated code resulted from developers and ChatGPT
- A publicly available dataset of developers and ChatGPT interactions, labelled with the prompt types and the final use cases of the generated code.

2 STUDY DESIGN

The main goal of this study is to gain valuable insights into how ChatGPT are now being utilized as a tool in real-world code practice, which can help direct future research in this domain. We focus on answering two key Research Questions (RQs). RQ1 seeks to investigate the characteristics of the interaction between the developers and LLMs. Specifically, in RQ1.1, we explore where exactly in the development process ChatGPT is being used. RQ1.2 studies the length of conversation between the developers and ChatGPT as the average number of prompt-response rounds needed to reach a satisfactory conclusion. And finally, RQ1.3 is designed to identify the common categories in which developers improve ChatGPT's output using different prompting strategies.

RQ2 targets to understand how and to what extent developers utilize generated code in production.

Kailun Jin, Chung-Yu Wang, Hung Viet Pham, Hadi Hemmati

2.1 DevGPT Dataset

To answer the mentioned RQs, we utilize the DevGPT [24] dataset. It is formed by searching with the keyword "https://chat.openai.com/share/" in the GitHub GraphQL API to identify mentions of shared links sourced from software development artifacts, such as source code, commits, issues, pull requests, discussions, and Hacker News threads. Eventually, DevGPT comprised 17,913 prompts and ChatGPT's responses, which encompass 11,751 code snippets. The dataset provides six snapshots at six specific points from July 24th to August 31st, 2023.

2.2 RQ1 Design

To answer RQ1, we followed two steps: (a) Automated data cleaning and (b) Manual labelling.

Automated data cleaning: The original DevGPT dataset comprised ChatGPT Sharing URLs from GitHub and Hacker News references. Since this study focuses on interactions between developers and ChatGPT on GitHub, we filter out conversations from Hacker News sources. DevGPT includes ChatGPT conversations linked from five GitHub artifacts: source **Code files**, **Commits**, **Issues**, **Pull requests**, and **Discussions**. The DevGPT dataset contains JSON files documenting the details of each ChatGPT interaction. Parsing the "Conversations" attribute in the JSON files allows us to extract the total number of conversation rounds for each ChatGPT interaction. However, during our analysis, we identified instances where conversations included irrelevant prompt-response rounds such as expressions of gratitude and greetings to ChatGPT. For instance, following a successful solution, developers frequently express their appreciation by stating, "Thank you, the approach worked", bringing the conversation to a close. Alternatively, they might commence the interaction with a greeting such as, "Hi, can you help me with the problem?". To eliminate such prompts, two co-authors conducted manual reviews of the conversations and excluded a total of 344 irrelevant conversation rounds from the 2,299 conversations related to code generation. To ensure the conversations were correctly reviewed, each conversation was reviewed by both authors and any disagreement was resolved with a discussion.

Manual labeling: To address RQ1.3, we looked into how developers use the prompts to have a conversation with ChatGPT for code generation and how they evolve their prompts to improve the outputs. We used an existing list of categories from a recent study [18] where the authors have looked into the same problem but within a small interview study.

In our case, due to the volume of data to label, we utilize the crowd-sourcing process to label each data sample (each conversation) using one of the labels [18] as shown in Table 2.

To run the crowd-sourcing, first, we developed a description of the task with clear instructions and examples based on the definitions above. The task is assigning each pair of generated code (accessed via the given ChatGPT sharing link) and changed source code (access via the given source Github link) to one of the 9 categories as shown in Table 2. Using crowd-sourcing methods, we distributed the 2,299 valid

data entries to volunteers whom we recruited from our local software engineering community. The crowdsourcing model leverages the experience and diversity of our local software engineering community to ensure the high-quality and professional classification of data. Each pair of data was reviewed by three developers and the majority was used as the label. If all three initial judgments were completely different, the opinions of another two developers were added. We continue adding two more opinions until a category gets at least two votes more than any other category.

2.3 RQ2 Design

To answer RQ2 we look into code generation conversations that are shared in Pull Requests. As shown in Table 1, there are 189 distinct conversations pertaining to code generation within the Pull Request category. To efficiently determine whether the generated code is implemented in the actual production system, we utilized the "View review changes" functionality for each merged Pull Request on GitHub. Consequently, we excluded 51 Pull Request records that were not merged into the primary project, resulting in 138 merged Pull Requests forming the basis for our data in terms of usage classification.

During our initial investigation, we found that some of the generated code snippets are used in the master branch (either directly as a copy, with some modifications, or integrated only into the instructional document), while a lot of them were not used at all. Thus we assign each generated code snippet with one of the following labels, manually (leveraging the View review changes functionality of GitHub) labeled by two co-authors (independently, and then double checked and discussed if disagreed until resolved):

- **Exact Match** denotes instances where developers employed the code exactly as generated by LLM without any modifications.
- **Modified Code** signifies cases where developers made slight alterations to certain sections of the generated code to align with the production code.
- **Document** encompasses situations where developers incorporated the generated code into instructional documents, such as README or test case files.
- **Supplementary Info** characterizes scenarios where developers did not directly employ the generated code but provided it as supplementary information or as a source of conceptual inspiration.

To verify the accuracy of the labels, each code snippet was reviewed by both authors and any disagreement was resolved with a discussion.

3 RESULT AND DISCUSSION

In this section, we will present our findings on RQ1 and 2.

3.1 RQ1: How do the developers interact with ChatGPT for code generation?

To investigate the interaction between developers and ChatGPT across different timestamps, we extracted and filtered all accessible and distinct conversations from each of the six

Table 1: The number of analyzed conversations between ChatGPT and developers.

GitHub categories	CG	Non-CG	Both	% of CG
Pull Request	189	79	268	70.5%
Issue	362	154	516	70.2%
Discussion	36	23	59	61.0%
Commit	660	10	670	**98.5%**
Code File	1052	958	**2010**	52.3%
Total	2299	1224	3523	65.3%

Table 2: The number of studied code generation conversations per prompt category, as defined by Shin at el[18]

Categories	Count
Request improvements	613
Request more description	353
Add specific instructions	351
Ask questions to find correct way	297
Add more context	256
Request examples	155
Request verification	93
Point mistake then request fix	112
Request another generation	69
Total	2299

snapshots in the DevGPT dataset. Parsing JSON files and choosing conversations with multiple instances of "Listof-Code" in ChatGPT's responses, we categorize a conversation as related to code generation (CG) if it contains at least one code snippet in ChatGPT's responses. Table 1 displays the number of conversations categorized as related to code generation (CG) and not related to code generation (Non-CG). The conversations are split into five GitHub categories (five rows) with a total number of analyzed conversations listed in the Total row. The five categories correspond to the sources of conversations (i.e., **Pull Request**, **Issue**, **Discussion**, **Commit**, and Source **Code file**. For each category, the number of CG, Non-CG, and both types of conversations along with the percentage of CG-related conversations are listed in the corresponding row.

RQ1.1: What is the distribution of conversations from different GitHub sources? Overall, we analyze 3523 conversations with 65.3% of them are CG-related. Our analysis shows that while, the majority of conversations occurred in the context of code files (2010), only about half (52.3%) of them are directly associated with code generation which is the lowest among all categories. In contrast, the Commit category exhibits a markedly high association with code generation with the majority of conversations (98.5%) involving some form of code snippet generation. Our manual analysis shows that, in code file conversations, developers often seek clarification on certain code snippets rather than asking for code generation. On the other hand, during Commit-related discussions, developers frequently ask ChatGPT to improve provided code snippets. A further explanation is mentioned in RQ1.2.

RQ1.2: How many prompt-response rounds does it take on average upon conclusion? To further analyze the characteristics of the conversations between developers and ChatGPT in a code-generating context, we investigate the number of prompt-response rounds required for each CG-related conversation to reach its conclusion. Due to the presence of irrelevant conversation rounds at times, we manually analyze each CG-related conversation and count the number of rounds required for the code-generation question to be satisfactorily answered by ChatGPT. Figure **??** shows the box plot that represents the distribution of the number of required prompt-response rounds for the five GitHub categories. The means are represented by the red stars and the medians are represented by the orange lines.

Figure **??** highlights that conversations in the Commit context lasted on average only 2.4 rounds while conversations

in the Code file context lasted on average significantly longer at 10.4 rounds.

RQ1.3: How do developers use ChatGPT to improve their code? As described in Section 2, to investigate how developers use ChatGPT to improve their code, we analyze the six checkpoints from DevGPT. We manually categorized the data based on the nine conversational prompting categories, as proposed by Shin et al. [18]. Table 2 shows the number of studied conversations per category.

Most of the conversations (more than a quarter) are about "**Request improvements**" while only a few are about "**Request examples**" or "**Request verification**". After consolidating the findings from RQ1.2, it was observed that 64.98% of conversations sourced from Commit focused on *Code Improvement*, including "request improvements", "Add specific instructions", "Add more context", "Request verification", "Point mistake then request fix" and "Request another generation". Additionally, 35.02% of conversations within Code files centred around *Code Clarification and Explanation*, such as "Request more description", "Ask questions to find correct way" and "request examples". The result explains that, when developers provide specific code snippets directly to ChatGPT for improvement (in the Commit context), the desirable outcomes are likely achieved quicker with fewer clarifications or explanations of the prompts. Conversely, more prompt-response rounds are needed to explain and clarify code functions in Code file conversations.

Table 2 shows that "**Request another generation**" conversation is rare. We suspect that since GPT-3.5 is based on transformer architecture, it will tend to "remember" recent conversations and "forget" older context[8]. This leads to additional generation requests to confuse ChatGPT into generating bad-quality code, hence the developers avoided this option.

3.2 RQ2: How helpful is the code generated by ChatGPT in assisting developers?

As described in Section 2, we categorize the conversation based on how the generated code snippets are used (if at all) in the master branch of the corresponding project. Figure 1 shows the proportions of conversation in each usage category.
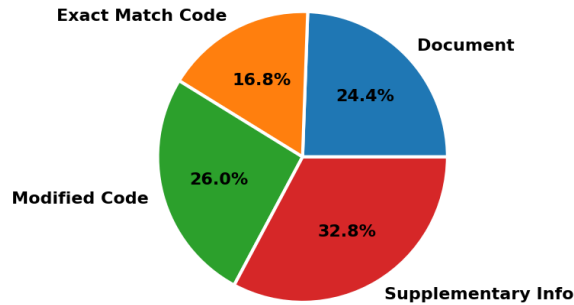
Kailun Jin, Chung-Yu Wang, Hung Viet Pham, Hadi Hemmati

Figure 1: The proportion of different generated code usages in real-world projects

As shown in Figure 1, in most cases, the generated code is **Supplementary Info** to developers, perhaps due to the inferior quality. For instance, a commit conversation indicates that the generated code is considered unhelpful, causing a notable slowdown instead of enhancing code execution speed. Sometimes (24.4%), the code snippets are used in instructional **Documents** or test cases. For instance, developers could request ChatGPT to generate a README file by providing a prompt that includes descriptions of the GitHub repository.

In 16.8% of conversations, the generated code snippets are **Exact Matches** to the code in the master branch of the corresponding projects. In such cases, the snippets are used exactly or with very trivial modifications. For instance, developers submit a prompt that encompasses a segment of the source code along with details about an exception they encountered. They request ChatGPT to identify and modify the source code. The generated code is subsequently employed directly in the source code without any further modifications.

In some other cases (26%), the generated code snippets are heavily **Modified** before being included in the source code of the corresponding projects. For instance, developers might rename variables or add exception handling before merging into the primary project.

3.3 Discussion

Survivorship bias poses a validity threat in the DevGPT conversations dataset, as it is constructed from retained conversations, potentially overlooking a broader spectrum of developer interactions. The filtering bias, where developers are less likely to include ChatGPT URLs in commit messages if the generated code's quality is too low. This may skew the dataset towards resolved discussions. Therefore, conclusions like "Commits often require fewer prompt-response rounds on average" and "a majority of conversations occurred in the context of code files," may be based on incomplete information. We acknowledge dataset limitations based on empirical experience during manual dataset processing, but lack of quantitative evidence. Future work might need to address this threat is essential for more robust conclusions.

4 RELATED WORK

Evaluation of LLMs in generating code: Recent improvements [21, 26] in LLMs' code generation ability has brought

attention to evaluating LLM-generated code from research communities. LLMs have been evaluated in terms of generated code correctness [1, 4, 12], robustness [26]. Prior work [5, 26] has also conducted human studies to evaluate how code generation supports developers in their tasks. These studies have demonstrated the potential of LLMs in code generation, however, they were all conducted in research settings. Hence the LLMs' effectiveness in real-world scenarios is inconclusive. [19]. This study focuses on accessing the code generation capabilities of LLMs using data scraped from an open-source platform dedicated to software development.

Interaction with LLMs regarding source code: Prior studies analyze the interaction of developers with LLMs to improve the usability [2] and reliability [15] of code generation. Prior studies discovered that LLMs reduce distractions during software development [11] and introduce fewer vulnerabilities [16]. However, studies have shown that both novices (e.g., students) [14] and experienced developers [17] can struggle to use LLMs on programming tasks. Our study provides some insight on all of the above aspects when LLMs are used in real-world development settings.

Code generation on software development platforms: StackOverflow and GitHub are the two most popular platforms used in software development for purposes such as communication, question-answering, and collaborative efforts [23]. Recently, LLMs have been used to generate code addressing problems raised on these platforms. [9, 26]. However, there have been limited prior studies that analyze the actual usage of such generated code in real-world workflow[7]. This study aims to bridge that gap and provide additional insight into how LLM code generation is integrated into software development.

5 CONCLUSION

Large language models such as ChatGPT have shown promises in code generation.

Our study reveals that while a majority of ChatGPT conversations within code files focus on discussions, only half are related to code generation. Commit-related interactions predominantly revolve around code improvement, with fewer prompt-response rounds. Developers primarily use ChatGPT for requesting improvements and tend to avoid additional code generation within the same conversation to prevent confusion. Notably, 32.8% of generated code is not used, emphasizing the need for further exploration of the practical utility of AI-generated code. These insights provide valuable considerations for refining AI-assisted development tools and enhancing collaboration between developers and AI systems. Much improvement is needed before LLMs become an integral part of modern software development.

6 ACKNOWLEDGEMENT

REFERENCES

[1] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732* (2021).

[2] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.

[3] Christian Bird, Denae Ford, Thomas Zimmermann, Nicole Forsgren, Eirini Kalliamvakou, Travis Lowdermilk, and Idan Gazit. 2023. Taking Flight with Copilot: Early Insights and Opportunities of AI-Powered Pair-Programming Tools. *Queue* 20, 6 (jan 2023), 35–57. https://doi.org/10.1145/3582083

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

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

[6] Luciano Floridi and Massimo Chiriatti. 2020. GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines* 30 (2020), 681–694.

[7] Mehdi Golzadeh, Tom Mens, Alexandre Decan, Eleni Constantinou, and Natarajan Chidambaram. 2022. Recognizing bot activity in collaborative software development. *IEEE Software* 39, 5 (Sept. 2022), 56–61. https://doi.org/10.1109/MS.2022.3178601

[8] Albert Gu, Caglar Gulcehre, Tom Le Paine, Matt Hoffman, and Razvan Pascanu. 2020. Improving the Gating Mechanism of Recurrent Neural Networks. arXiv:1910.09890 [cs.NE]

[9] Jae Yong Lee, Sungmin Kang, Juyeon Yoon, and Shin Yoo. 2023. The GitHub Recent Bugs Dataset for Evaluating LLM-based Debugging Applications. *arXiv preprint arXiv:2310.13229* (2023).

[10] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161* (2023).

[11] Jenny T Liang, Chenyang Yang, and Brad A Myers. 2023. Understanding the Usability of AI Programming Assistants. *arXiv preprint arXiv:2303.17125* (2023).

[12] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *arXiv preprint arXiv:2305.01210* (2023).

[13] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. arXiv:2102.04664 [cs.SE]

[14] James Prather, Brent N Reeves, Paul Denny, Brett A Becker, Juho Leinonen, Andrew Luxton-Reilly, Garrett Powell, James Finnie-Ansley, and Eddie Antonio Santos. 2023. " It's Weird That it Knows What I Want": Usability and Interactions with Copilot for Novice Programmers. *arXiv preprint arXiv:2304.02491* (2023).

[15] Steven I Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D Weisz. 2023. The programmer's assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. 491–514.

[16] Gustavo Sandoval, Hammond Pearce, Teo Nys, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg. 2022. Security implications of large language model code assistants: A user study. *arXiv preprint arXiv:2208.09727* (2022).

[17] Advait Sarkar, Andrew D Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. 2022. What is it like to program with artificial intelligence? *arXiv preprint arXiv:2208.06213* (2022).

[18] Jiho Shin, Clark Tang, Tahmineh Mohati, Maleknaz Nayebi, Song Wang, and Hadi Hemmati. 2023. Prompt Engineering or Fine Tuning: An Empirical Assessment of Large Language Models in Automated Software Engineering Tasks. arXiv:2310.10508 [cs.SE]

[19] Giriprasad Sridhara, Sourav Mazumdar, et al. 2023. ChatGPT: A Study on its Utility for Ubiquitous Software Engineering Tasks. *arXiv preprint arXiv:2305.16837* (2023).

[20] Lewis Tunstall, Leandro Von Werra, and Thomas Wolf. 2022. *Natural Language Processing with Transformers, Revised Edition.* "O'Reilly Media, Inc.".

[21] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*. 1–7.

[22] Junjie Wang, Yuchao Huang, Chunyang Chen, Zhe Liu, Song Wang, and Qing Wang. 2023. Software testing with large language model: Survey, landscape, and vision. *arXiv preprint arXiv:2307.07221* (2023).

[23] Shaowei Wang, David Lo, and Lingxiao Jiang. 2013. An empirical study on developer interactions in stackoverflow. In *Proceedings of the 28th annual ACM symposium on applied computing*. 1019–1024.

[24] 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)*.

[25] Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 1–10.

[26] Li Zhong and Zilong Wang. 2023. A study on robustness and reliability of large language model code generation. *arXiv preprint arXiv:2308.10335* (2023).