Analyzing Developer Use of ChatGPT Generated Code in Open Source GitHub Projects

Balreet Grewal, Wentao Lu, Sarah Nadi, and Cor-Paul Bezemer
{balreet,wlu4,nadi,bezemer}@ualberta.ca
University of Alberta
Edmonton, Alberta, Canada

ABSTRACT

The rapid development of large language models such as ChatGPT have made them particularly useful to developers in generating code snippets for their projects. To understand how ChatGPT's generated code is leveraged by developers, we conducted an empirical study of 3,044 ChatGPT-generated code snippets integrated within GitHub projects. A median of 54% of the generated lines of code is found in the project's code and this code typically remains unchanged once added. The modifications of the 76 code snippets that changed in a subsequent commit, consisted of minor functionality changes and code reorganizations that were made within a day. Our findings offer insights that help drive the development of AI-assisted programming tools. We highlight the importance of making changes in ChatGPT code before integrating it into a project.

ACM Reference Format:
Balreet Grewal, Wentao Lu, Sarah Nadi, and Cor-Paul Bezemer. 2024. Analyzing Developer Use of ChatGPT Generated Code in Open Source GitHub Projects. In *Proceedings of MSR '24: Proceedings of the 21st International Conference on Mining Software Repositories (MSR 2024).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The rise of Large Language Models (LLMs), especially the Generative Pre-trained Transformers (GPT) series by OpenAI, has revealed a new era in the field of Artificial Intelligence (AI) and software development. ChatGPT [15] is OpenAI's chatbot that utilizes the GPT family of models. Users can chat with the model to receive context-aware, human-like responses. A notable area where ChatGPT has shown significant promise is in software development. Developers have been increasingly relying on the model to generate code snippets [9, 23, 28, 29]. This trend has sparked interest in the academic community, leading to numerous studies focusing on the correctness, and applicability of ChatGPT-generated code in real-world scenarios [11, 12, 14, 18, 20, 27].

Despite the growing body of research on ChatGPT and other large language models, there remains a gap in our understanding of the longevity (how long it takes for code to change in a project) and evolution (the nature of the subsequent changes) of AI-generated code in software development contexts. Our study narrows this gap by conducting an empirical analysis of how ChatGPT-generated code is integrated within GitHub projects. By examining the integration, modification, and deletion of this code, we seek to provide insights into AI-aided software development. Specifically, we focus on the following research questions:

RQ1: How much of ChatGPT's generated code is changed by developers? We analyzed the integration of ChatGPT's generated code into GitHub projects by examining the percentage of code used and the likelihood of further changes. We found that a median of 54% of lines of code from a ChatGPT-generated code snippet is embedded in a GitHub project. Moreover, making a subsequent change to the integrated code within a GitHub project is uncommon.

RQ2: How long does ChatGPT-generated code stay unchanged in a project? To understand when a ChatGPT-generated code snippet might change, we explored for how long the code snippets have been integrated in GitHub projects, and how long until a change is made to the code snippet. A ChatGPT-generated code snippet remains in a project for a median of 89 days, and if the code snippet is changed, it is typically done so within a day.

RQ3: What types of changes do developers make to ChatGPT's generated code? We manually categorized the types of modifications made to the generated code snippets after their integration into a project. We observed that of the 76 code snippets that changed in a subsequent commit, 14 code snippets were deleted. The remaining generated code snippets undergo modifications, often involving minor functionality changes, reorganization of the code or name refinements.

The main contributions of our paper are as follows:

- The first empirical study on how code generated by ChatGPT is integrated into software projects on GitHub.
- A categorization of the types of changes made to ChatGPT-generated code in GitHub repositories.

2 METHODOLOGY

This section outlines our methodology for analyzing code snippets generated by ChatGPT and their integration into open source projects. Figure 1 gives an overview of our methodology.

2.1 Selecting conversations with code snippets

We utilized the DevGPT dataset [1, 25], which features developer and ChatGPT conversations as explicitly linked by developers in GitHub projects. The dataset includes commits, pull-requests, files, issues, or discussions that connect to a ChatGPT conversation. In cases where the recorded conversation leads to generated code, the

MSR 2024, April 15–16, 2024, Lisbon, Portugal
© 2024 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

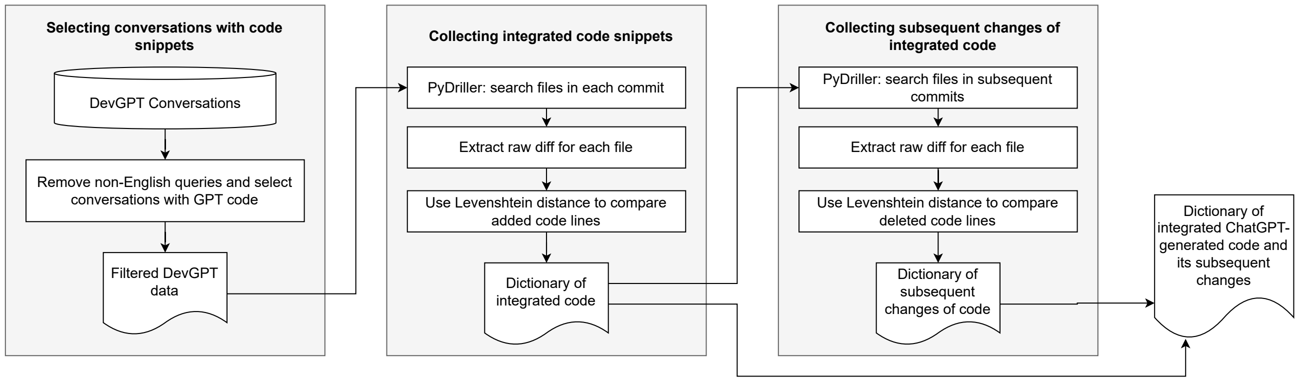
Selecting conversations with code snippets
DevGPT Conversations
Remove non-English queries and select conversations with GPT code
Filtered DevGPT data
Collecting integrated code snippets
PyDriller: search files in each commit
Extract raw diff for each file
Use Levenshtein distance to compare added code lines
Dictionary of integrated code
Collecting subsequent changes of integrated code
PyDriller: search files in subsequent commits
Extract raw diff for each file
Use Levenshtein distance to compare deleted code lines
Dictionary of subsequent changes of code
Dictionary of integrated ChatGPT-generated code and its subsequent changes

Figure 1: Overview diagram of our methodology

resulting code snippets are stored in the dataset. A linked conversation does not always contain a ChatGPT-generated code snippet; as a result, a single conversation may include anywhere from 0 to multiple code snippets. We specifically focus on ChatGPT conversations linked in commits within GitHub projects to ensure there is at least one code snippet.

2.2 Cleaning the data

In this step, we remove all non-English conversations from the dataset using the *langdetect* [19] Python library. We also removed conversations that were linked with four inaccessible repositories. After cleaning the data, we were left with 9,610 code snippets from 56 GitHub repositories.

2.3 Collecting integrated code snippets

Since the DevGPT [25] data only shows the conversations between developers and ChatGPT, it is unknown how much and which of the code generated by ChatGPT during a conversation is integrated into a project. We utilized Git diff with the tool Pydriller [21] to identify which lines of a generated code snippet were integrated into a project's main branch in GitHub. The Git diff output shows the different lines between two commits. At this stage, for each generated code snippet, we only collected the added lines from the Git diff output (as we focus on code that was added into the project). Once we identify a code line from ChatGPT that matches an added line in the Git diff, we consider this as a matched code pair. However, sometimes, an exact match of two lines is not sufficient to identify matched code pairs. For example, in Listing 1, the developer [5] obtained a code snippet from ChatGPT to configure virtual machines. The developer reduced the assigned number of CPUs from 4 to 2 when integrating the code into the project. We consider this a match as well because the vast majority of the line was generated by ChatGPT.

To include such pairs, we calculate the Levenshtein distance with Python's *Levenshtein* library [8] for string comparisons of each code line. The Levenshtein distance (Lv distance) measures the difference between two strings. Thus, rather than relying on exact string matching, the Lv distance allows for a degree of flexibility to account for minor variations in code integration, such as discrepancies in assigned numbers. The matched strings are recorded

```
1 Vagrant.configure(
      VAGRANTFILE_API_VERSION
      ) do |config|
2   ...
3     v.memory = 1024
4     v.cpus = 4
```

```
1 Vagrant.configure(
      VAGRANTFILE_API_VERSION
      ) do |config|
2   ...
3     v.memory = 1024
4     v.cpus = 2
```

Listing 1: Example of a ChatGPT-generated code snippet (left) that was integrated in a GitHub project (right). All lines were matched (including line 4) with the Lv distance set to 2.

in the output dictionary as illustrated in Figure 1. A range of Lv distance thresholds were empirically tested, and a threshold of 2 was chosen as optimal.

Through this method, we identified that 3,044 out of 9,610 ChatGPT-generated code snippets were at least partially integrated into GitHub projects. We use the 3,044 code snippets in further analysis.

2.4 Collecting subsequent changes of integrated code snippets

After the collection of 3,044 code snippets that were integrated into GitHub projects, we further analyzed the evolution of their code lines. We used PyDriller to trace commit histories over the main branch only, focusing on subsequent changes to the same files that contain those integrated code snippets from ChatGPT. Attention was given to deleted code lines, since any modifications to a code line are represented as deletions in Git diff files. For example, the subsequent commit for the code in Listing 1 shows that it was removed the next day. We recorded any modified lines into the output as illustrated in Figure 1. We identified 76 code snippets that were changed in a subsequent commit.

3 RESULTS

In this section, we answer our three research questions.

3.1 RQ 1: How much of ChatGPT's generated code is changed by developers?

Motivation: We analyze how much of ChatGPT's generated code is changed before it is integrated into a project. Our insights can

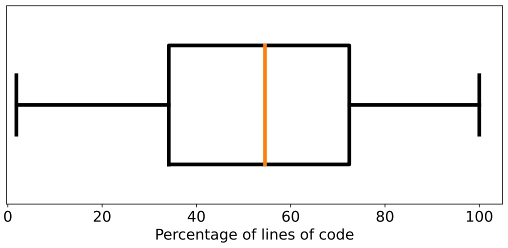

Figure 2: Boxplot of the percentage of ChatGPT's generated code in a single code snippet that is found in GitHub projects.

help drive the development of tools for AI-assisted programming (e.g., such as those that suggest hints on how to change a generated code snippet).

Approach: We analyzed the number of lines of code (LOC), including comments, generated by ChatGPT in a given code snippet, as well as the number of LOC that match in GitHub projects.

To calculate the percentage of lines in a GitHub project derived from a single code snippet generated by ChatGPT, we use the following formula:

$$\text{\% Code Snippet Used} = \frac{\text{\# LOC integrated into the project}}{\text{\# LOC generated by ChatGPT}}$$

Findings: **A median of 54% of the generated ChatGPT code snippet is embedded in a GitHub project**. Figure 2 shows a boxplot of the percentage of ChatGPT-generated lines of code used in GitHub. Only 319 of the 3,044 studied ChatGPT code snippets are found entirely in GitHub project files. The median number of lines of code added to GitHub from a single code snippet is 28, whereas the median number of lines of code provided by ChatGPT in a single code snippet is 52. These numbers show that in the vast majority of cases, developers need to edit ChatGPT code before including it in their projects.

Once added to a GitHub project, it is uncommon for code from ChatGPT to be changed again. Only 76 (2.5%) of the 3,044 integrated ChatGPT code snippets were changed in a subsequent commit (with a median change of 1 line of code). Only 5 of the 76 code snippets integrated ChatGPT's generated code as-is.

Only 10% of the code snippets are copied from ChatGPT as-is. However, once a code snippet from ChatGPT is integrated in a GitHub project, only 2.5% change again with a median change of 1 line.

3.2 RQ 2: How long does ChatGPT-generated code stay unchanged in a project?

Motivation: By investigating how long until developers require changes to ChatGPT-generated code, we seek to gain insights into the longevity of ChatGPT-generated code in software development.

Approach: We analyze the number of days between the integration of the ChatGPT code and its first change (or our data collection date if the code did not change).

Findings: **ChatGPT code snippets have been integrated in GitHub projects for a median of 89 days**. Figure 3 shows the number of days since ChatGPT-generated code that was not changed in a subsequent commit has been added to GitHub. This finding indicates the sustained presence of AI-generated code snippets in software projects.

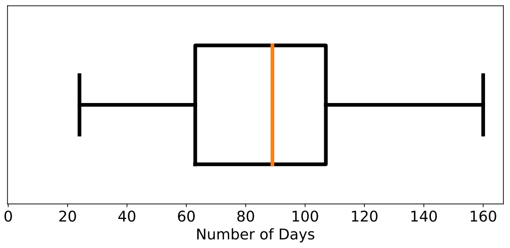

Figure 3: Boxplot of the number of days since the ChatGPT-generated code snippet (that was not changed in a subsequent commit) has been in a GitHub project.

If ChatGPT code changes in a subsequent commit, it typically does so in less than a day from when it was first added. If code is changed within GitHub after being added, it is done within the same day with the exception of a 4 outliers that take longer (between 1 and 12 days).

ChatGPT-generated code copied into GitHub projects has been integrated for a median of 89 days. If a change is required to the integrated code it is usually done within a day.

3.3 RQ 3: What types of changes do developers make to ChatGPT's generated code?

Motivation: We categorize the types of changes that developers made to ChatGPT-generated code to help developers who use AI-assistance better understand which types of code changes are common. This better understanding can in turn drive the development of better tools for supporting AI-assisted programming.

Approach: We perform open card sorting on the 76 code snippets that were changed after their initial commit on GitHub. Card sorting [22] is a common technique used to organize information and create mental models. In our case, we make use of open card sorting to categorize the types of changes made to ChatGPT code when changed in GitHub. Each of the authors performed card sorting on the code snippets individually. Then the authors sat down to discuss any discrepancies and reach a consensus.

Findings: **18% of the code snippets changed in a subsequent commit are deleted, whereas the other 82% of code snippets are modified.** Table 1 shows that 14 of the 76 (18%) ChatGPT-generated code snippets that are changed in a subsequent commit are deleted from the GitHub project; the rest of the code snippets were modified. The modifications consisted of minor functionality changes, code reorganization and naming refinements.

Minor functionality changes (36%) consist of changes that have only a minor impact on the functionality. For example, a declaration such as `var picks = this.getPicks();` was changed to `const pick = this.getPicks();` [3] We found 27 code snippets that required minor functionality changes such as changing the type of a variable or a small part of a used regular expression. This shows that ChatGPT-generated code may require small changes to be properly integrated with the project's existing code.

Type of change	Description	# of changes	% of changes
Minor functionality	Modifications that have only a minor impact on the functionality	27	36%
Reorganization	Relocating code	25	33%
Deletion	Complete deletion of the code snippet	14	18%
Naming refinement	Changing the name of elements such as functions, variables, and other identifiers	10	13%

Table 1: The types of changes made to ChatGPT-generated code in a subsequent commit

Code reorganization (33%) happens when specific lines of ChatGPT's generated code are relocated to a different section within the same file or to another file entirely. Notably, 24 out of the 25 occurrences of code reorganizations involved moving the code into a different function within the same file.

Naming refinement (13%) occurs when a name assigned by ChatGPT to an element in the code was unclear or too long. For example, a function initially named `replaceSpotifyPlayerWithYouTubePlayer()` was changed by developers to `replaceSpotifyPlayer()` [4]. From the ChatGPT generated code snippets we analyze, 10 of the code snippets required name refinement.

In a subsequent commit, 82% of code snippets undergo modifications such as minor functionality changes, code reorganization and name refinements. Only 14 of the 76 analyzed code snippets were deleted.

4 THREATS TO VALIDITY

Internal validity: A potential threat to the validity of our study is that the data we analyze is recent. As such, some developers have not yet needed to change ChatGPT's generated code. From Figure 3 it seems that most of ChatGPT's generated code has been in its respective repository for a considerable amount of time, but we encourage future studies to examine if our findings hold over a longer period of time.

Further, the Lv distance threshold used in our study is fixed to 2. Higher or lower thresholds may produce more matched code pairs depending on dataset and programming language used. For our study, we found the value 2 optimal to incorporate any changes but leave out minute details such as brackets.

External validity: In our study we rely on developers to link ChatGPT conversations to their GitHub commits. Thus, our findings are only based on the commits that explicitly state the use of ChatGPT code and may not be generalizable to repositories that use ChatGPT generated code without stating it.

Further, our results are solely based on open-source projects on GitHub; therefore, our results may not generalize to projects on other platforms or closed-source projects.

5 RELATED WORK

The body of research on the code generated by large language models (LLMs) [9, 23, 28, 29] has grown rapidly since the release of popular large language models such as GPT-3.5 and GPT-4. Ensuring correctness of code generation is imperative as the code generated by large language models has been employed to write code for various software development tasks such as automatic program repair [16, 17, 24], and unit tests [20, 27]. Some studies [14, 18] have analyzed ChatGPT's ability at solving Leetcode [2] problems, revealing that ChatGPT out-performs novice programmers and has a success rate of 72% of generating correct solutions. To aid in checking the correctness of code generated by LLMs, Liu et al. [11] built a new evaluation framework for benchmarking code on LLMs to examine their true correctness.

Further, studies have explored the quality of code generated by LLMs [6, 7, 13, 26]. For example, Liu et al. [12] studied the code quality of 4,066 pieces of generated code from ChatGPT, and ChatGPT's self-debugging capability. The authors found that code quality is an issue in code generated by ChatGPT and that debugging is heavily influenced by feedback given by users.

While existing literature focuses on characteristics (e.g., quality and correctness) of code generated by LLMs, such as ChatGPT, our work is the first to analyze how developers make use of code generated by ChatGPT within their software projects. We aim to understand how the generated code snippets are used by developers and if they require changes once integrated.

6 ETHICAL IMPLICATIONS

When conducting research, it is imperative to be conscientious of various ethical implications. First, privacy concerns can emerge as we analyze data within GitHub without explicit consent. Further, proper credits should be given to the authors of their repositories when discussing them in written work. We ensure that all code used from GitHub projects is linked within our references. Finally, to ensure that our report is transparent and reproducible, we made our replication package available at [10].

7 CONCLUSION

In this paper, we analyze the integration of 3,044 ChatGPT generated code snippets in GitHub projects. To augment the DevGPT data (consisting of ChatGPT and developer conversations), we search through commit histories in GitHub to trace the evolution of ChatGPT's generated code. We find that a median of 54% of ChatGPT's generated code is added to GitHub and it is uncommon for this code to be changed again. Of the 76 code snippets that eventually are changed a subsequent time in GitHub the change is typically made within a day and consists of a median of 1 line of code. Typically, the changes involve modifications such as minor functionality changes, name refinement, and code reorganization. Our findings offer insights that can inform the advancement of AI-assisted programming tools.

REFERENCES

[1] 2023. DevGPT: Studying Developer-ChatGPT Conversations. https://github.com/NAIST-SE/DevGPT/tree/35d906d957026f3db282b19dcc5074e399010725.

[2] 2023. *LeetCode*. https://leetcode.com

[3] 2023. Refactoring for interactivity. (#8). https://github.com/hoshotakamoto/banzukesurfing/commit/90e1d68ddc8d3a2caa076ee4d423484bf0a742f3. Accessed: December 1, 2023.

[4] 2023. simpler approach to replace all spotify embedded players. https://github.com/OKinane/spotify-to-youtube-chrome-extension/commit/5d8f6f8f5c2457348f5739888b5d5bd4260ac8cb. Accessed: December 1, 2023.

[5] 2023. Update Vagrantfile. https://github.com/fabricesemti80/work.ansible-prometheus-stack/commit/96c4f63bbdba293001c540f663337a0dec41e71c. Accessed: December 1, 2023.

[6] Naser Al Madi. 2023. How Readable is Model-Generated Code? Examining Readability and Visual Inspection of GitHub Copilot. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 205, 5 pages. https://doi.org/10.1145/3551349.3560438

[7] Carlos Dantas, Adriano Rocha, and Marcelo Maia. 2023. Assessing the Readability of ChatGPT Code Snippet Recommendations: A Comparative Study. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering* (Campo Grande, Brazil) *(SBES '23)*. Association for Computing Machinery, New York, NY, USA, 283–292. https://doi.org/10.1145/3613372.3613413

[8] Wido de Vries. 2017. *python-Levenshtein*. https://pypi.org/project/python-Levenshtein/

[9] 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)*. 1469–1481. https://doi.org/10.1109/ICSE48619.2023.00128

[10] Balreet Grewal, Wentao Lu, Sarah Nadi, and Cor-Paul Bezemer. 2023. *Analyzing Developer Use of ChatGPT Generated Code in Open Source GitHub Projects Replication Package*. https://doi.org/10.6084/m9.figshare.24792507

[11] 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:2305.01210 [cs.SE]

[12] 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:2307.12596 [cs.SE]

[13] Zhijie Liu, Yutian Tang, Xiapu Luo, Yuming Zhou, and Liang Feng Zhang. 2023. No Need to Lift a Finger Anymore? Assessing the Quality of Code Generation by ChatGPT. arXiv:2308.04838 [cs.SE]

[14] Nascimento Nathalia, Alencar Paulo, and Cowan Donald. 2023. Artificial Intelligence vs. Software Engineers: An Empirical Study on Performance and Efficiency Using ChatGPT. In *Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering* (Las Vegas, NV, USA) *(CASCON '23)*. IBM Corp., USA, 24–33.

[15] OpenAI. 2023. *ChatGPT*. https://www.openai.com/

[16] Julian Aron Prenner, Hlib Babii, and Romain Robbes. 2022. Can OpenAI's Codex Fix Bugs? An Evaluation on QuixBugs. In *Proceedings of the Third International Workshop on Automated Program Repair* (Pittsburgh, Pennsylvania) *(APR '22)*. Association for Computing Machinery, New York, NY, USA, 69–75. https://doi.org/10.1145/3524459.3527351

[17] Francisco Ribeiro. 2023. Large Language Models for Automated Program Repair. In *Companion Proceedings of the 2023 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity* (Cascais, Portugal) *(SPLASH 2023)*. Association for Computing Machinery, New York, NY, USA, 7–9. https://doi.org/10.1145/3618305.3623587

[18] Fardin Ahsan Sakib, Saadat Hasan Khan, and A. H. M. Rezaul Karim. 2023. Extending the Frontier of ChatGPT: Code Generation and Debugging. arXiv:2307.08260 [cs.SE]

[19] Nakatani Shuyo. 2011. *langdetect*. https://pypi.org/project/langdetect/

[20] Mohammed Latif Siddiq, Joanna C. S. Santos, Ridwanul Hasan Tanvir, Noshin Ulfat, Fahmid Al Rifat, and Vinicius Carvalho Lopes. 2023. An Empirical Study of Using Large Language Models for Unit Test Generation. arXiv:2305.00418 [cs.SE]

[21] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. PyDriller: Python Framework for Mining Software Repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Lake Buena Vista, FL, USA) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 908–911. https://doi.org/10.1145/3236024.3264598

[22] Donna Spencer and Todd Warfel. 2004. Card sorting: a definitive guide. *Boxes and arrows* 2, 2004 (2004), 1–23.

[23] 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 *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, Article 332, 7 pages. https://doi.org/10.1145/3491101.3519665

[24] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated program repair in the era of large pre-trained language models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE 2023). Association for Computing Machinery*.

[25] 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)*.

[26] 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.

[27] Zhiqiang Yuan, Yiling Lou, Mingwei Liu, Shiji Ding, Kaixin Wang, Yixuan Chen, and Xin Peng. 2023. No More Manual Tests? Evaluating and Improving ChatGPT for Unit Test Generation. arXiv:2305.04207 [cs.SE]

[28] Quanjun Zhang, Tongke Zhang, Juan Zhai, Chunrong Fang, Bowen Yu, Weisong Sun, and Zhenyu Chen. 2023. A Critical Review of Large Language Model on Software Engineering: An Example from ChatGPT and Automated Program Repair. arXiv:2310.08879 [cs.SE]

[29] Li Zhong and Zilong Wang. 2023. Can ChatGPT replace StackOverflow? A Study on Robustness and Reliability of Large Language Model Code Generation. arXiv:2308.10335 [cs.CL]