Bachelor's Thesis Computing Science

Radboud University Nijmegen

Developer-ChatGPT Conversations: Analyzing Developer Prompts

Author:
Yiğit Batuhan Yiğin
s1082159

First supervisor/assessor:
dr. Mairieli Wessel

Second assessor:
dr. Bin Lin

October 22, 2024

Abstract

As large language models such as ChatGPT become more popular, developers started integrating these tools for software development tasks, often sharing their conversations on platforms like GitHub. Understanding how developers interact with ChatGPT in the context of GitHub pull requests and issues can provide important insights into the dynamics of these interactions. In this research, we analyze the characteristics of the conversations between developers and ChatGPT in the context of GitHub pull requests and issues using the DevGPT dataset. We specifically investigate common words, inclusion of code blocks, types of questions, follow-up prompts and sentiments. Our results reveal that 43% of pull request conversations contain code blocks, this ratio is 29% for issue conversations. Code blocks mostly appear in the first prompt of the conversation. Furthermore, closed questions are used more frequently than open questions for both pull requests and issues. Developers provide follow-up prompt 65% of the time. Finally, positive sentiment was the most common, followed by neutral sentiment.

Contents

Chapter 1

Introduction

ChatGPT is OpenAI's groundbreaking large language model-based chatbot, published on November 30, 2022. By January 2023, it reached more than 100 million users [4]. Even though a chatbot's main function is to mimic human conversations, ChatGPT is highly versatile.

A vast majority of software developers often resort to online resources for software-related tasks e.g. bug fixing, learning new technologies, understanding of various concepts, etc. These online resources include official documentation and Q&A platforms such as Stack Overflow to seek help from other developers [7]. However, the emergence of ChatGPT has demonstrated a potential shift in this paradigm. Recognizing the value of this technology, many software developers have incorporated it into their daily workflows [2]. One of the primary ways developers use ChatGPT is to quickly generate lines of code, which helps developers to skip mundane parts of writing code. Furthermore, it is used for debugging by providing the already-written code and information related to the errors that are occurring. Developers also use it to seek assistance with problem-solving, learning new technologies, and many other tasks.

A new and powerful feature of ChatGPT is that conversations can be shared with others through a link. As a result, ChatGPT is being used on collaborative developer platforms such as GitHub, where multiple developers work on a project together [20]. For instance, a software development team might be working on a project together and one of the team members encounters a problem in their task. The team member then solves the problem with the help of ChatGPT. However, the solution might not be entirely correct, this is where link sharing feature becomes crucial. The team members can attach this link to the solution they have provided, and other team members can review and provide more insight into the usability of this solution. Sharing the conversation history takes place in various GitHub project sections such as source file, commit, issue, pull request, and discussion. This feature makes it possible for multiple people to review the output

taken from ChatGPT. As a result, the feature for reviewing the conversation with multiple people helps to prevent from using low-quality solutions and to increase the efficiency of team collaboration.

Despite ChatGPT's popularity, there has not been extensive research into the quality and usability of its responses to software-related queries [7]. Furthermore, little is known about how developers use ChatGPT, for instance, what kind of questions they pose and to what degree the conversations are dynamic [20]. Therefore, this research aims to fulfill the aforementioned knowledge gaps by analyzing the DevGPT data set [9]. Furthermore, this research aims to help future researchers by providing valuable insights into how ChatGPT can assist better in software development tasks, enhance code quality and improve productivity. In addition, our findings can guide engineers in future model improvements for software development use. The dataset contains data related to prompts and ChatGPT's responses coupled up with software development artifacts such as raw source files, commits, issues, pull requests or in discussions [20]. In our research, we solely focus on two artifacts, namely pull requests and issues. We address the research questions below by analyzing the data on pull requests and issues so that the efficiency of ChatGPT use for software development can be enhanced.

Research question:

- RQ1: What are the characteristics of developer interactions with ChatGPT in the context of pull requests and issues?

We answer RQ1 with the following sub-questions:

- SQ1: What are the most common words developers use in ChatGPT conversations shared in pull requests/issues?
- SQ2: How frequently do developers include code blocks in their conversations within pull requests/issues?
- SQ3: When does a code block appear within a conversation?
- SQ4: Are open or closed questions more frequent in conversations about pull requests/issues?
- SQ5: How often do developers provide follow-up prompts?
- SQ6: What sentiments are common in conversations related to pull requests/issues?

We answer SQ1 by preprocessing prompt data, tokenizing, and then using word frequency analysis.

For SQ2, we manually detect and replace code blocks within prompts with a placeholder such as "[CODEBLOCK]". If any prompt in conversations contains this placeholder, we then count that conversation as having a code block.

SQ3 is answered similarly to SQ2. Instead of counting, we identify the specific prompt in which the first code block appears within each conversation.

For SQ4, we first categorize each prompt containing specific question words (e.g. why, how, and so on) into open and closed categories. Then we count the occurrences of both question types.

For SQ5, we categorize conversations based on whether they have one prompt or more than one. Then we get the results by counting each conversation.

For SQ6, we perform sentiment analysis on the prompts using the VADER Sentiment Analysis library. We then determine the sentiment of a conversation by taking the majority sentiment of prompts. The result is then the count of conversations of each sentiment (e.g. neutral, positive, or negative).

Chapter 2

Background

In this chapter, we first give more context about the key concepts related to this thesis. Furthermore, we discuss already existing related research.

2.1 ChatGPT

ChatGPT is an interactive large language model (LLM) in the form of a chatbot developed and launched by OpenAI on November 30, 2022. It is based on the Generative Pre-trained Transformer (GPT) architecture designed to handle natural language processing (NLP) tasks, including text generation and language comprehension [15]. It has shown powerful functions on various language understanding and text generation tasks such as multilingual machine translation, code debugging, story writing, admitting mistakes, and even rejecting inappropriate requests according to the official statement [19]. Unlike previous chatbots, ChatGPT can remember the context of the conversation, which makes it possible to have continuous conversations [5].

Software developers are utilizing ChatGPT for various tasks such as code generation or helping with making decisions [2]. It can speed up the coding process by providing suggestions and generating large code blocks in a few seconds [14]. An important and one of the newest features of ChatGPT is the ability to share conversations with others via links. This allows others to see every prompt and answer of a conversation. This feature is particularly useful in collaborative software development, where developers can review AI-generated content and reduce potential problems.

2.2 GitHub Issues

GitHub Issues are reports repository users create when a problem or unexpected behavior occurs [3]. Issues are straightforward to work with and flexible, they can be used for various things such as to track the work in

a repository, give or receive feedback, collaborate, and communicate with others who are also working in the same repository by leaving comments under an issue. Issues also provide an overview of the project's progress.

Figure 2.1: An example GitHub issue

Figure 2.2 shows an example GitHub issue. Here, the issue title is "Bug in the Header React component". The round green box indicates that the issue is still open and needs to be resolved. There are two comments on this issue: one from the owner of the issue (yeetzpxoci) added when creating the issue, and another one from the user "curious_ant". The first comment in this issue has a reference to another issue which is in the same repository, this helps to establish a connection between related issues.

2.3 GitHub Pull Requests

Another important feature that GitHub has is a mechanism called pull requests. They are proposals to merge a feature branch of a repository into the main branch. A collaborator first forks the repository, creates a feature branch, makes changes while continuously committing these changes, and creates a pull request to request the changes to be merged into the main branch [16]. Finally, the collaborators decide whether they want to accept this new feature and merge it into the main branch or decline.

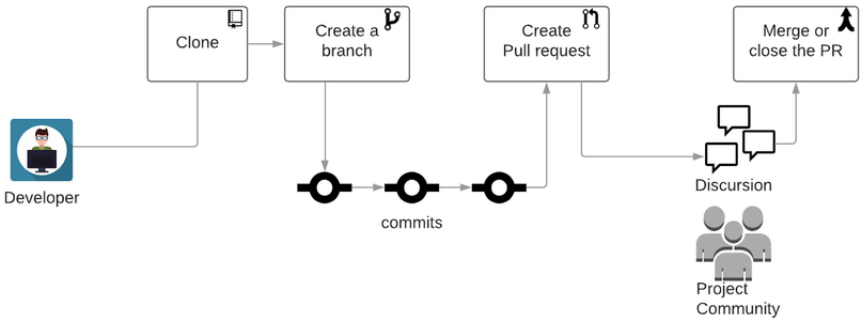


Figure 2.2: Workflow of GitHub Pull requests

Various factors contribute to the success and failure of a pull request. Some of these factors are the programming language of a project, the application domain, the project age, the developers and how experienced they are, and so on [12].

Chapter 3

Related Work

With ChatGPT rapidly gaining popularity, many research papers on this cutting-edge technology are published daily. In this section, we inspect and analyze some of the related research already done.

AlOmar et al.[1] tried to analyze the usage of ChatGPT in code refactoring utilizing the DevGPT dataset. They inspected the conversations between developers and ChatGPT to understand how developers identify parts of code that can be improved and how ChatGPT addresses the refactoring. They used an approach that relies on text mining the conversations related to refactoring. Their result revealed that the conversations mostly involve generic and specific phrases. In addition, refactoring requests made by developers are often generic, on the other hand, ChatGPT frequently incorporates the refactoring purpose. They found textual patterns that describe developers' refactoring requests, the most frequently occurring refactoring operations included "extract*", "mov*" and "renam*". Furthermore, they identified the quality attributes ChatGPT considers when describing refactoring. They specified three categories for the quality attributes: internal, external and code smells. They discovered that ChatGPT emphasizes specific coding practices such as dependency injection, naming conventions, unit tests, and software design patterns. Finally, they found out that most developers copy-paste code fragments along with a text that describes how to fix the issue.

Champa et al. [2] performed a comprehensive analysis of the usage of ChatGPT in software development. They also utilized the DevGPT dataset. Identified types of tasks software developers seek assistance for with ChatGPT. The most common task types included code quality management, commit issue resolution, generation of documentation, implementation of a new feature, and so on. Python was the most common language for the first two types of tasks, while for the last it was Shell. In addition, they also found out which task types tend to result in more or less effective help from ChatGPT. The most efficient task types were software development man-

agement and optimization, and new feature implementation, while the least efficient was documentation generation. They inspected if the initial prompt had any effect on the efficiency of a task's completion. They inspected the correlation between the length of the conversations and the prompt quality using metrics such as mutual information, distance correlation, and Spearman correlation. The prompt quality includes three metrics: readability score, sentence complexity, and grammar errors. The results indicated that the initial prompt does not affect the efficiency of completing a task with the help of ChatGPT.

Jin et al. [6] Conducted an empirical analysis of conversations found in the DevGPT dataset. They focused on how developers interact with ChatGPT in terms of code generation and how helpful the generated code is in helping developers. They found out that 65.3% of the conversations within GitHub sources were related to code generation. They manually labeled the data using a crowdsourcing process. Most of the conversations were requests for improvements, adding descriptions, etc. while only a few were about requesting examples or verification. They found out that providing code snippets immediately to ChatGPT for improvement makes getting the desired outcome more efficient. Regarding being helpful, 32.8% percent of the ChatGPT generated code is not used. This shows the need for further examination of the practical usability of ChatGPT-generated code.

Rabbi et al. [11] performed a quantitative analysis using the DevGPT dataset. They assessed the quality and security issues found in the Python code snippets that were generated by ChatGPT. They categorize code snippets into two: ChatGPT-generated(generated by ChatGPT from scratch) and ChatGPT-modified(modified user-provided code). They use four metrics for the comparative quantitative analysis, both for code quality issues and security issues. The results showed that the most frequent code quality issue was errors followed by convention violations, warnings, and refactoring suggestions. Developer-provided code had a lower amount of errors, convention violations, and refactoring suggestions compared to ChatGPT-modified code. In terms of security issues, they checked different CWEs (Common Weakness Enumeration). But in contrast to the code quality issues, both ChatGPT-generated and ChatGPT-modified code equally included security vulnerabilities.

Yang et al. [21] is a systematic literature review about how non-functional properties other than the accuracy of large language models for code (LLM4Code) are evaluated and improved. These properties include robustness, security, privacy, explainability, efficiency, and usability. In total, 146 relevant papers were identified. Despite the success of LLM4Code in vulnerability detection and code generation, research highlights their flaws such as low robustness, lack of explainability, and insecure code production. Through the systematic literature review, they categorize the challenges and opportunities for improving LLM4Code into three different views: data-centric, human-centric,

and system-centric. The data-centric view is concerned with the training data LLMs use, it emphasizes the training dataset quality. Human-centric view on the other hand focuses on the continuous interaction of LLMs with humans. Adoption of a human-centric view can improve the usability and developers' well-being. Lastly, the system-centric view aims to enhance the integration of LLM4Code into broader systems where it is considered an essential part of the overall system. This view is more real-life-oriented than the previous two. Some of the research opportunities identified include identifying high-quality training data, factors that affect usability, and building attack-resistant systems.

Researchers have tried to evaluate the effectiveness of ChatGPT as a tool for software engineers [10]. This evaluation is divided into three parts: evaluation of the code refactored by ChatGPT, evaluation of a simple application created by ChatGPT, and evaluation of machine learning code created by ChatGPT. For the refactoring task, the researchers provided ChatGPT with smelly C# code (a characteristic of code that indicates a problem with the fundamental design principles). The results indicate that ChatGPT improved the structure, and readability and applied the best practices in software development. The refactored code is well-structured and easily maintainable. This indicates that ChatGPT can be a really helpful tool for software engineers in enhancing the overall quality of code. However, the code taken from ChatGPT may contain mistakes and not be fully correct.

ChatGPT also created a fully working tic-tac-toe game that can be played against a smart bot. It implemented the bot by making use of game theory. It can also generate fully working code for machine learning. It created a machine learning code to predict if user input indicates diabetes. The code generated by ChatGPT was almost indistinguishable from the code generated by humans in terms of prediction accuracy. This shows ChatGPT's potential as a useful tool in the domain of machine learning.

This paper concludes that ChatGPT can be a really powerful tool in software development for various tasks. However, the code generated by ChatGPT might consist of problems or unexpected behavior, therefore, these solutions must be taken with a grain of salt. Even if the generated code is correct, it can still not really understand anything. Because of this, contextual problems might also arise. In addition, ChatGPT's reliance on training data potentially results in biases or weaknesses in its responses.

[22] This study focuses on generator tools that utilize LLM's abilities. These tools consist of ChatGPT, GitHub Copilot, and Amazon CodeWhisperer. They analyze the quality of generated code for each tool regarding validity, correctness, security, reliability, and maintainability using the HumanEval [1] dataset. The results demonstrated that ChatGPT was the most successful code-generation tool among all three. ChatGPT was the winner

[1] `https://github.com/openai/human-eval/`

in terms of validity and correctness. In terms of code security, there were no differences between the tools. Finally, the reliability and maintainability aspects of the tools can not be compared, as most vulnerabilities and code smells are unique to the tool and vary in severity.

[13] conducted a comparative analysis on successful (merged) and unsuccessful (open or closed) pull requests under different aspects. They looked at pull requests under the eight most common technical issues. The success rate of the merge pull requests was rather low, with only 7.76% on average. Furthermore, they inspected pull requests for different programming languages such as Ruby, Java, and JavaScript. The results were identical to those obtained in the analysis of technical issues. The average number of unsuccessful pull requests was significantly higher than the successful pull request number. The study suggests that this might be potentially related to excessive forking; however, more future research is needed on language-specific factors affecting pull request success. Finally, projects in the IDE, Framework, and Client Apps domains demonstrated a higher success rate of pull requests compared to the Networking, Database, Statistics, and Library domains.

Another study tried to verify whether the rejection rate of the pull request is related to the quality of the code [8]. They used a tool called PMD, which can be used for static code analysis. However, they were unable to find any correlation between code quality and pull request acceptance.

Finally, research also tried to identify and present prompt patterns to solve common problems when using LLMs for software engineering-related tasks [18]. They classified different patterns into different categories according to the type of problem they were trying to solve. These categories include Requirements Elicitation, Code Quality, System Design, and Simulation and Refactoring. In each category, various patterns specify the structure of the prompts. According to the paper, the patterns can help reduce the errors LLMs make while performing software engineering tasks by providing. They concluded the research by mentioning that the capabilities of LLMs like ChatGPT are not fully understood, and human expertise is still needed to leverage ChatGPT for software engineering tasks.

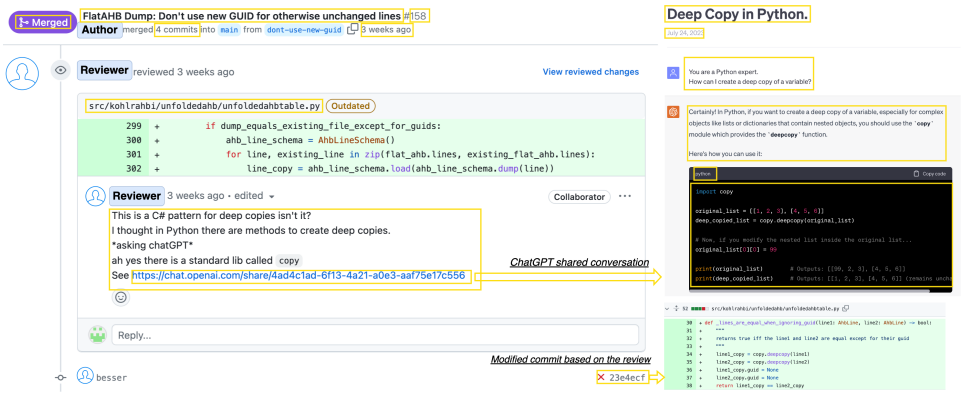
Chapter 4

Methodology

4.1 Dataset

The dataset [9] consists of links to ChatGPT conversations between software developers and ChatGPT. The data is categorized into software development artifacts such as source code, commits, issues, pull requests, and Hacker News threads. The data for an artifact is stored in a JSON file and is called X_sharings.json, where X is the artifact name. A data entry in an artifact is called a "ChatgptSharing" (code sharing, commit sharing, etc.) and has additional information related to the entry itself, such as the source of the entry, author, and so on.

Figure 4.1: An example entry in the context of GitHub pull request



There are 6 different snapshots (datasets) collected at 6 different times, the last one being the most recent one. Each snapshot contains the data from the previous snapshot and new additional data; thus, we chose to work with the most recent snapshot. As this research is only concerned with GitHub issues and pull requests, we decided to focus only on the data of

these two artifacts. The structure of the relevant data is shown in the tables 4.1, 4.2, 4.3, 4.4 and 4.5 respectively.

Attribute	**Description**
`Type`	Source type
`URL`	URL to the mentioned source
`Author`	Author who introduced this mention
`RepoName`	Name of the repository that contains this issue
`RepoLanguage`	Primary programming language of the repository (can be null)
`Number`	Issue number
`Title`	Title of the issue
`Body`	Description of the issue
`AuthorAt`	When the author created this issue
`ClosedAt`	When this issue was closed (can be null)
`UpdatedAt`	When the latest update occurred
`State`	The state of the issue (OPEN or CLOSED)
`ChatgptSharing`	List of ChatGPT link mentions

Table 4.1: Structure of the GitHub issue sharing data

Attribute	Description
`Type`	Source type
`URL`	URL to the mentioned source
`Author`	Author who introduced this mention
`RepoName`	Name of the repository that contains this pull request
`RepoLanguage`	Primary programming language of the repository (can be null)
`Number`	Pull request number
`Title`	Title of the pull request
`Body`	Description of the pull request
`CreatedAt`	When the author created this pull request
`ClosedAt`	When this pull request was closed (can be null)
`MergedAt`	When this pull request was merged (can be null)
`UpdatedAt`	When the latest update occurred
`State`	The state of the pull request (OPEN, CLOSED, MERGED)
`Additions`	Number of lines added in this pull request
`Deletions`	Number of lines deleted in this pull request
`ChangedFiles`	Number of files changed in this pull request
`CommitsTotalCount`	Number of commits included in this pull request
`CommitShas`	List of commit Shas included in this pull request
`ChatgptSharing`	List of ChatGPT link mentions

Table 4.2: Structure of the GitHub Pull Request sharing data

Attribute	Description
`URL`	URL to the shared ChatGPT links
`Mention`	Attributes about mention. Refer to the `Mention` structure for details
`Status`	HTTP response status when accessing this URL
`DateOfConversation`	Date when this conversation occurred
`DateOfAccess`	Date when we accessed this URL
`NumberOfPrompts`	Number of prompts in this conversation
`TokensOfPrompts`	Number of tokens of prompts in this conversation
`TokensOfAnswers`	Tokens of answers in this conversation
`Model`	Version of the model used in this conversation
`Conversations`	List of conversations. Refer to the `Conversations` structure for details
`HTMLContent`	HTML content from the shared ChatGPT link

Table 4.3: Structure of the ChatgptSharing data

Attribute	Description
`MentionedURL`	URL to the mentioned source
`MentionedProperty`	What kind of property was mentioned in this shared ChatGPT link
`MentionedAuthor`	Who mentioned this shared ChatGPT link
`MentionedText`	The context when this shared ChatGPT link was mentioned
`MentionedPath`	Where the comment was added in files for reference of shared ChatGPT links in review threads of GitHub pull requests (only exists when referenced in review threads of pull requests)
`MentionedIsAnswer`	Whether this comment is marked as an answer in discussion (only exists when referenced in comments of discussion)
`MentionedUpvoteCount`	Number of upvotes that this comment has received (only exists when referenced in comments of discussion)

Table 4.4: Structure of the Mention data

Attribute	Description
`Prompt`	Prompt that user inputted
`Answer`	Answer that ChatGPT generated
`ListOfCode`	Code list from the answer (includes code type and content)

Table 4.5: Structure of the Conversations data

Statistic	Pull Requests (PR)	Issues
Total Number of Sharings	152	311
Total Number of Prompts	723	1259
Total Number of Code Blocks	130	215
Mean Number of Code Blocks per Sharing	0.855263	0.691318

Table 4.6: Summary Statistics of ChatGPT Sharings in Pull Requests and Issues

4.1.1 Data Preprocessing and Preparation

We used Jupyter Notebook with Python 3.10 for reading, preprocessing, and analyzing the data. The pull request and issue-sharing data were read from corresponding CSV files and converted into pandas DataFrames using the pandas library.

The prompts contained various code blocks, that can affect our data analysis, as these code blocks mostly consist of non-linguistic words. We had to manually replace these code blocks by reviewing all the prompts in our dataset as there was no specific formatting of the code blocks for every prompt. In a conversation, we replaced the code blocks with placeholders "CODEBLOCK0", "CODEBLOCK1" and so on. We identified code blocks by looking at specific phrases such as "here is the code:", and "this is the code:" or keywords used in code such as "import", "if", "else" and so on. Moreover, we only included sharings that contained only a single ChatGPT conversation to make our analysis more consistent.

We created additional features for all sharings that are useful for the data analysis:

- **Prompts**: List of prompts used in the corresponding sharing. Punctuation marks were removed, and all letters were converted to lowercase.
- **Language**: Represents the language of the conversations found in the sharing. We used langid for categorizing the prompts. The language of a conversation is then the majority language of the prompts.
- **NumberOfCodeBlocks**: Total number of code blocks included in the corresponding sharing.
- **FirstIndexOfCodeBlock**: Index of the prompt where a code block is encountered for the first time.
- **QuestionTypes**: List of question types that correspond to prompts respectively. We have come up with two question types, namely open and closed. Open questions are questions that contain question words that the answer is not simply "yes" or "no" ("how", and "why" are

such question words). Closed questions, on the other hand, have an answer of "yes" or "no", these have words such as "is", "can", "does", and so on. To categorize the prompts, we checked if a prompt contained any words corresponding to one of the two question types.

To ensure consistent results, we filtered out non-English conversations, as the linguistic structure and availability of language-specific libraries (e.g., VADER Sentiment Analysis) could skew results.

4.2 Type of Question

We defined two categories for the types of questions: open and closed. Treude et al. [17] did a similar categorization by question types. However, they only had the category for how-to questions and other categories were related to the topics of the questions. For both pull requests and issue sharings, prompts were categorized by identifying the presence of specific keywords. For open questions, we checked if the prompt contained the words "how" or "why". For closed questions, we checked if the prompt contained any of the words "is", "our", "do", "does", "can", "will", "would", or "should".

After categorization, each sharing was assigned a list of question types corresponding to its prompts. This list was then one-hot encoded to convert the categorical data into a numerical format. The one-hot encoded data was combined with the state column from the original dataset.

State	closed	open	other
OPEN	1	1	0
OPEN	1	0	0
OPEN	13	1	2
OPEN	2	1	1
OPEN	24	1	5

Table 4.7: Example result of the categorization

An example resulting data is presented in Table 4.7. Each row corresponds to a sharing, and the columns represent different types of questions. The cells indicate the number of prompts that contain that type of question. For example, in the last row, the pull request state for the sharing is "OPEN," and the sharing includes 24 prompts with closed questions, 1 prompt with an open question, and 5 prompts with other question types.

4.3 Follow-up Prompts

To investigate follow-up prompts, we labeled each sharing based on the presence of follow-up prompts. Sharing that included conversations with only a single prompt was labeled as "true," indicating no follow-up prompts. Conversely, sharings that contained conversations with more than one prompt were labeled as "false". In addition, we look into the number of follow-up prompts.

4.4 Sentiment Analysis

We applied the VADER Sentiment Analysis library to classify each prompt into one of three categories: Positive, Negative, or Neutral. For each sharing, we determined the overall sentiment based on the most frequent sentiment category within its prompts. For example, if a conversation contained three prompts with two labeled as Negative and one as Neutral, the overall sentiment for that sharing would be classified as Negative.

Chapter 5

Results

5.1 Most Common Words

Word	Count
file	170
use	148
code	118
run	115
test	111
using	108
want	104
like	90
name	89
user	72

Table 5.1: Top 10 Most Common Words in Pull Request Sharings

Word	Count
file	328
line	230
using	193
use	192
code	187
return	140
get	159
would	151
module	144
import	135

Table 5.2: Top 10 Most Frequent Words in Issue Sharings

The word "string" occurred almost 2000 times. After checking the prompts that included "string", we have seen that they do not make logical sense, but rather gibberish prompts. Therefore, we decided to ignore the word "string" from the result and focus on other results.

The word "file" is the most common word used for pull requests and issue sharing. File operations such as writing or reading from a file are common tasks in programming projects. Developers might also begin their prompts using the word "file", e.g. "I have this code in my test.js file..."

"test" is another word that is common within pull request sharings. As pull requests also include test-related changes, it makes sense that the word

”test” is commonplace.

”line” is a high-frequency word in issue sharings but not in pull request sharings. GitHub issues are used commonly for reporting bugs and developers might use the word ”line” to point out bugs in the code. For example: ”There is a semi-colon missing on line 35”. The same explanation can be said for the word ”error”. ”import” and ”module” are both related to importing different modules or dependencies in projects, which are prone to errors, this explains why they occur frequently within issue sharings.

Words such as ”use”, ”using”, ”want”, ”like” and ”please” are all high-frequency words in both pull request and issue sharings (even though the last three are not in the top 10 for issue sharings, they are still in the top 15). These words are commonly used and not specific to certain tasks.

SQ1: What are the most common words developers use in ChatGPT conversations shared in pull requests/issues?

For pull requests, the most common words include ”string”, ”file”, ”use”, ”run” and ”code”. While, for issues words such as ”file”, ”line”, ”return”, and ”import” are included. For more details, refer to the tables 5.1 and 5.2.

5.2 Code Blocks

5.2.1 Pull Requests

Number of Code Blocks	Count of Conversations
0	88
1	37
2	18
3	4
4	3
22	1
11	1

Table 5.3: Number of conversations in pull request sharings per number of code blocks

Around 57% of the conversations do not contain any code blocks 5.3. This might indicate that developers mostly use ChatGPT for knowledge retrieval rather than direct requests with their existing code.

Number of Code Blocks	Count of Prompts
0	658
1	150
2	6
3	1

Table 5.4: Number of prompts in pull request sharings per number of code blocks

The majority of prompts included in pull request sharings do not contain any code blocks 5.4. This is because most of the time, developers just include the code block in one simple prompt and follow with other requests regarding this code block. A more interesting approach is to see when these code blocks are first introduced in a conversation and how often conversations include any code blocks at all, that is shown below.

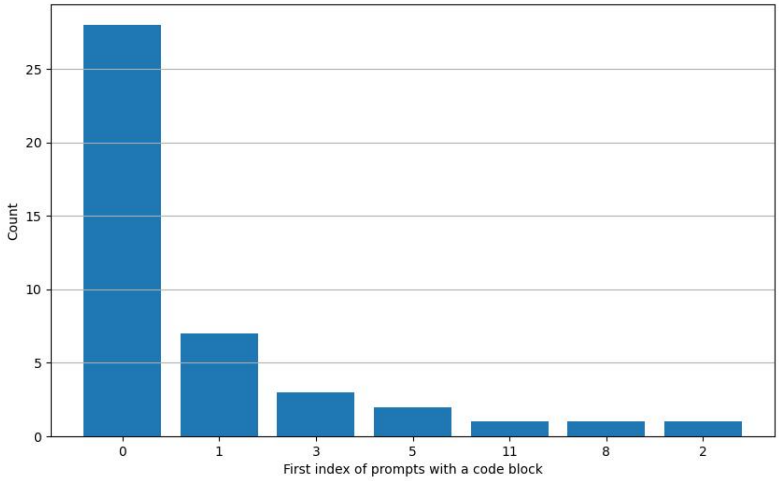


Figure 5.1: Histogram showing the distribution of the first index of prompts with a code block

Figure 5.1 shows the distribution of the first index of prompts that contain a code block. For example, in a conversation, if the first prompt contains a code block, this would be included with a 0 in figure 5.1. Most of the time, code blocks were included in the first prompt. As we are only interested in conversations with multiple prompts, conversations that contain only 1 prompt were not included. It can be seen from the resulting histogram that in most conversations, code blocks were introduced immedi-

ately. This indicates that developers first provide their code to ChatGPT for context, and then they provide follow-up requests.

5.2.2 Issues

Number of Code Blocks	Count of Conversations
0	206
1	70
2	18
3	8
4	5
14	2
5	1
6	1
7	1
8	1
11	1
13	1

Table 5.5: Number of conversations in issue sharings per number of code blocks

With 65%, most of the conversations in issues do not contain any code blocks. Compared to pull request sharings, conversations in issues are less likely to include code blocks. This can be explained by pull requests being directly related to code changes while issues are not necessarily about code changes but can be more general.

Number of Code Blocks	Count of Prompts
0	1114
1	191
2	7
3	2
4	1

Table 5.6: Number of prompts in issue sharings per number of code blocks

Similar to the pull request sharings, a high percentage of the prompts (around 88%) do not contain any code blocks. This is again due to develop-

ers including the code block in a single prompt and then providing multiple follow-up requests.

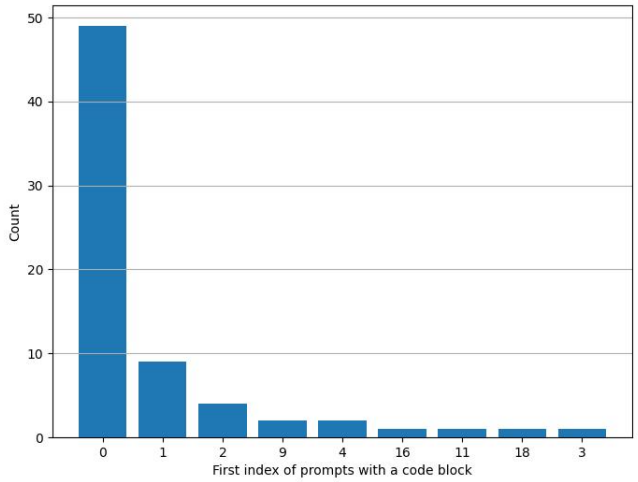


Figure 5.2: Histogram showing the distribution of the first index of prompts with a code block

Again, similarly to the pull request sharings, most code blocks were introduced immediately in a multi-prompt conversation. This again indicates that developers first provide their code for context, and then the follow-up requests.

SQ2: How frequently do developers include code blocks in their prompts within pull requests/issues?

For conversations related to pull requests, 43% contain code blocks. For issues, the percentage is 35%.

SQ3: When does a code block appear in a prompt?

For conversations related to pull requests, 65% of the time code blocks were introduced in the first prompt of a conversation. This percentage is 70% for issues.

5.3 Types of Questions

5.3.1 Pull Requests

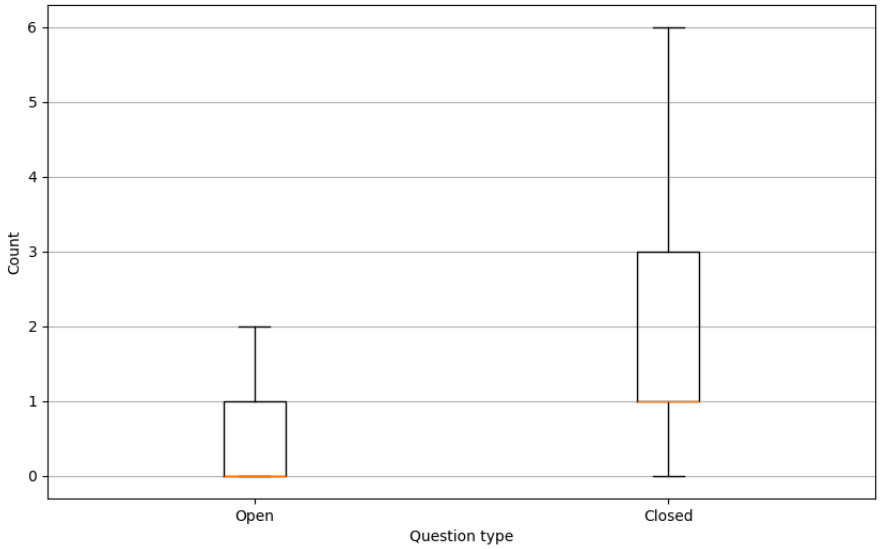


Figure 5.3: Box plot showing the distribution of the number of questions per question type in pull requests

The box plot 5.3 shows that closed questions are in the majority. As closed questions have a median of 1, we can say that most conversations contained in pull requests contain closed questions. On the other hand, open questions have a median of 0, it can be inferred that most conversations do not include open questions. In addition, closed questions have a broader range compared to open questions.

5.3.2 Issues

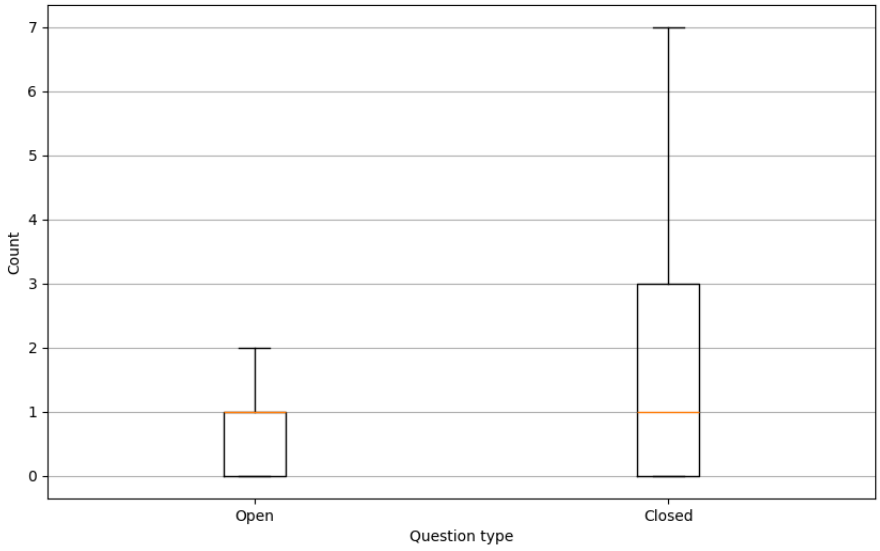


Figure 5.4: Box plot showing the distribution of the number of questions per question type in issues

In the case of issues, closed questions appear more compared to open ones. They also have a higher variety than open questions. Open questions seem to have a more uniform frequency of occurrence.

SQ4: Are open or closed questions more frequent within conversations related to pull requests/issues?

Closed questions are more frequent in both conversations related to pull requests and issues.

5.4 Follow-up Prompts

5.4.1 Pull Requests

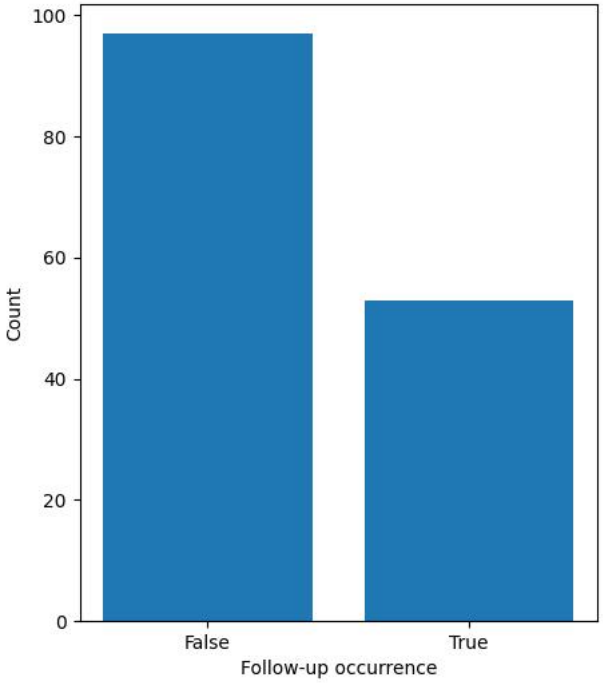


Figure 5.5: Count of pull request sharings with follow-up prompts

Around 65% of pull request sharings contain conversations with follow-up prompts. This indicates that developers usually keep asking additional questions after initial prompts in the context of pull requests.

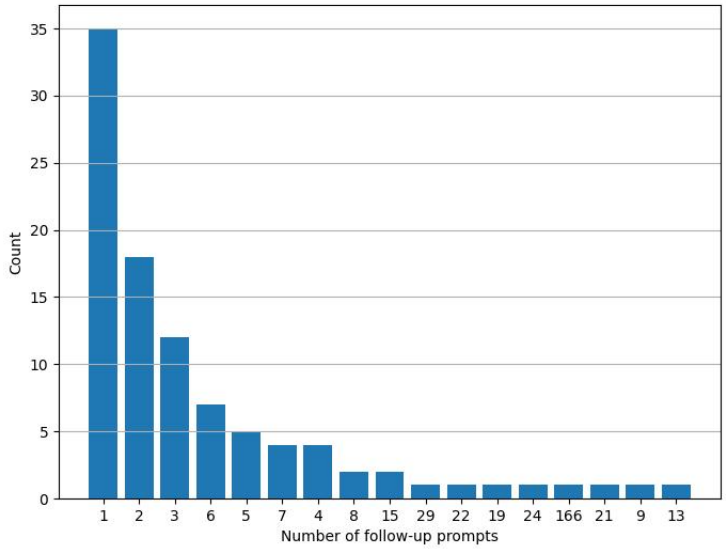


Figure 5.6: Number of follow-up prompts for pull requests

In pull requests, developers often provide only a single follow-up prompt before ending the conversation. As the number of follow-up prompts increases, the frequency drops sharply, indicating that extended conversations involving 4 or more prompts are uncommon in the context of pull requests.

5.4.2 Issues

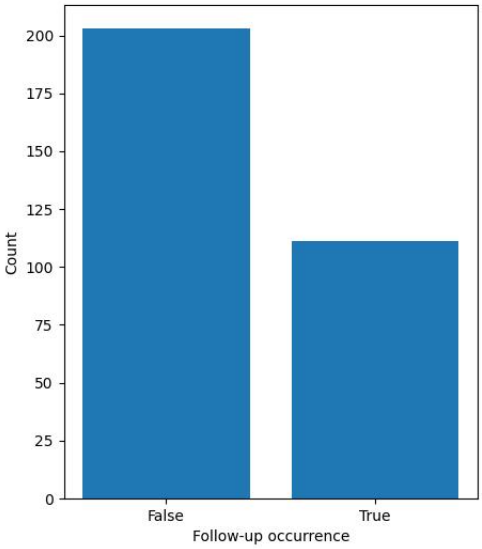


Figure 5.7: Count of issue sharings with follow-up prompts

Almost the same outcome can be seen for the issue sharings, this is most likely a coincidence. Around 65% of issue sharings contain follow-up prompts.

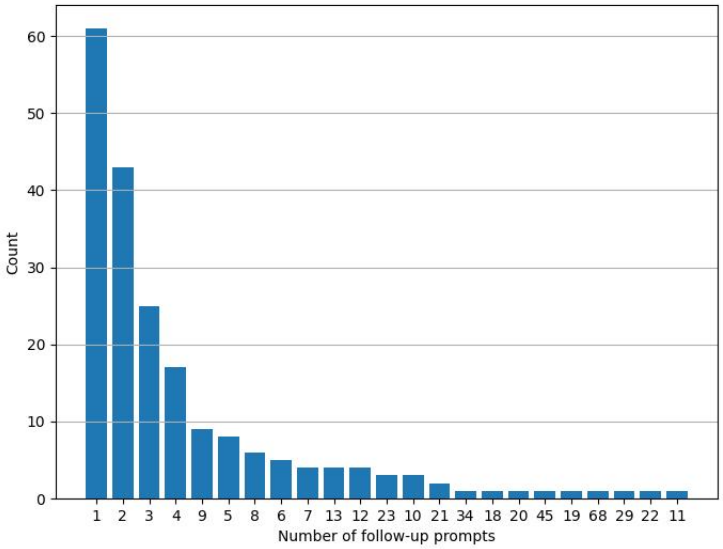


Figure 5.8: Number of follow-up prompts for issues

Similarly to pull request sharings, single follow-up prompt is the most popular in issues. However, two follow-up prompts are much higher compared to pull request sharings.

SQ5: How often do developers provide follow-up prompts?

Developers include follow-up prompts in 65% of the conversations found both in pull request and issues.

5.5 Sentiment

5.5.1 Pull requests

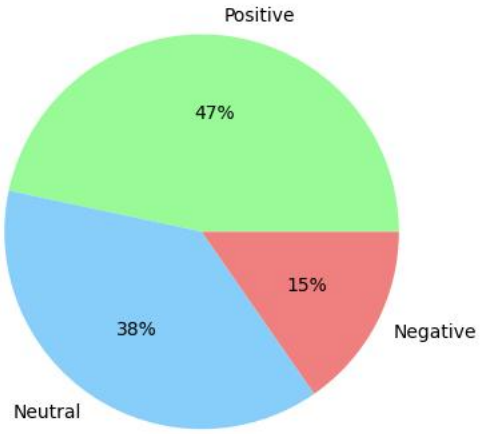


Figure 5.9: Pie chart of pull request sharing sentiment

Most of the pull request sharings were categorized as Positive and Neutral. The positive sentiment can be explained by the usage of the words like "please" or "thanks". "Please" was one of the top common words from section 6.1, both within pull requests and issue sharings, this explains the high percentage of positive sentiments. Furthermore, developers often voice their requests or questions neutrally without imposing any explicit emotions. Words that might indicate negative sentiment are really rare within the prompts, which explains the low percentage of negative sentiment.

5.5.2 Issues

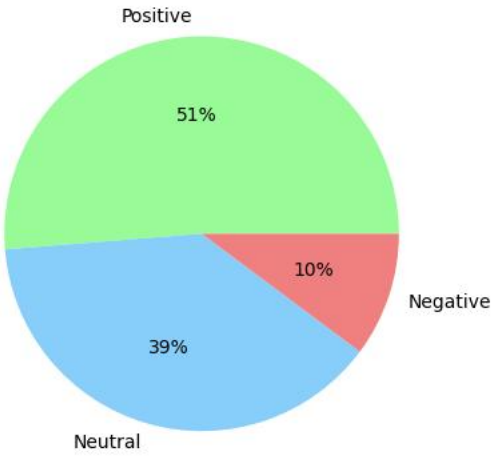


Figure 5.10: Pie chart of issue sharing sentiment

Similarly, issue sharings have a high percentage of positive sentiments. The percentage of negative sentiment is lower compared to pull requests.

SQ6: What sentiments are common in conversations related to pull requests/issues?

Positive sentiment is the most common, followed by neutral sentiment in both issues and pull requests. Negative sentiment is only a small percentage of all for both pull requests and issues, with issues having a smaller percentage.

Chapter 6

Discussion

In this chapter, we present the findings of our research, offering a detailed analysis of the results and their implications. Furthermore, we provide insights and recommendations for future research to further enhance the practical application of ChatGPT in developer interactions.

The most common words in conversations varied across the type of GitHub artifact, namely, pull requests and issues. Conversations regarding pull requests tend to include terms such as "file", "code" and "run". This result aligns with the focus of pull requests on the manipulation of code and execution. On the other hand, conversations about issues frequently included terms like "line," "module," and "error," indicating a focus on debugging. These findings are in line with AlOmar et al. [1] who also presented that developers often utilize ChatGPT for code manipulation and debugging.

Results showed that code blocks were found more frequently in conversations related to pull requests (43%) than issues (35%). This aligns with pull requests being closely related to code changes within projects. Yet, more than half of conversations related to pull requests did not contain any code blocks. This might result in ChatGPT giving lower or irrelevant answers, as ChatGPT doesn't have contextual clues from code blocks. We recommend future researchers to focus on this, as not enough research is available. Furthermore, code blocks are typically introduced within the first prompt of conversations (65% of the time for pull requests and 70% for issues). This suggests that developers often seek immediate help with their code snippets. Jin et al. [6] also underlined that providing code snippets directly streamlines getting help from ChatGPT.

The dominance of closed questions over open questions in both pull requests and issue-related conversations suggests that developers primarily use ChatGPT for confirmation rather than broader questions.

Follow-up prompts were frequently used by developers. This indicates that the developers mostly participate in continuous conversations with

ChatGPT, refining or providing more context to their initial prompt. Due to this continuous nature of conversations, developers might put less emphasis on providing high-quality initial prompts. This can also explain the results of the study [2] et al. which found out the initial prompt's quality did not impact task completion. In addition, most times, developers provided 3 or less follow-up prompts. This behavior may suggest that developers typically prefer short, productive interactions with ChatGPT and will discontinue the conversation or try a different strategy if the initial follow-ups do not lead to satisfactory results. We recommend that future researchers focus more on overall conversation rather than only initial prompts.

Positive sentiment is the most prevalent sentiment within conversations, both for pull requests and issues. This suggests that developers often use a friendly approach or achieve successful outcomes in their interactions with ChatGPT, reflecting satisfaction or encouragement during their problem-solving processes. Neutral sentiment is the second most frequent sentiment within conversations, both for pull requests and issues. This suggests that developers mostly remain objective and instructional, not showing emotions. This aligns with the developer's main goal of solving problems or knowledge retrieval. The relatively small proportion of negative sentiment, especially in issues, might suggest occasional frustrations that can arise during technical problem resolution. This pattern aligns with the findings of [11] et al., who observed that while ChatGPT often provides beneficial code modifications, it can also introduce errors or shortcomings, contributing to occasional disappointment within the conversation.

Chapter 7

Conclusions

Our data analysis revealed interesting trends of the usage of ChatGPT among developers. The most common words used in pull requests emphasize

Inclusion of code blocks were surprisingly lower than expected, especially for pull requests (43%) as they are closely related to code changes. In addition, most developers included the code blocks in their first prompt.

Questions asked by developers were mostly categorized as closed questions. This trend highlights a more task-focused and goal-oriented approach in their use of ChatGPT.

Follow-up prompts were common in developer interactions, suggesting that developers often engage in iterative dialogue with ChatGPT to refine their previous prompts.

The dominance of positive sentiment reflects the positive attitude and/or satisfactory outcomes of developers' interactions with ChatGPT. Following this, neutral sentiment indicates the practical and solution-oriented nature of developers.

Bibliography

[1] Eman Abdullah AlOmar, Anushkrishna Venkatakrishnan, Mohamed Wiem Mkaouer, Christian D. Newman, and Ali Ouni. How to refactor this code? an exploratory study on developer-chatgpt refactoring conversations. In *MSR*, pages 202–206. ACM, 2024.

[2] Arifa I. Champa, Md. Fazle Rabbi, Costain Nachuma, and Minhaz F. Zibran. Chatgpt in action: Analyzing its use in software development. In *MSR*, pages 182–186. ACM, 2024.

[3] Akash Balasaheb Dhasade, Akhila Sri Manasa Venigalla, and Sridhar Chimalakonda. Towards prioritizing github issues. In *ISEC*, pages 18:1–18:5. ACM, 2020.

[4] Krystal Hu. Chatgpt sets record for fastest-growing user base - analyst note, 2023.

[5] Wenxiang Jiao, Wenxuan Wang, Jen-tse Huang, Xing Wang, and Zhaopeng Tu. Is chatgpt A good translator? A preliminary study. *CoRR*, abs/2301.08745, 2023.

[6] Kailun Jin, Chung-Yu Wang, Hung Viet Pham, and Hadi Hemmati. Can chatgpt support developers? an empirical evaluation of large language models for code generation. In *MSR*, pages 167–171. ACM, 2024.

[7] Samia Kabir, David N. Udo-Imeh, Bonan Kou, and Tianyi Zhang. Who answers it better? an in-depth analysis of chatgpt and stack overflow answers to software engineering questions. *CoRR*, abs/2308.02312, 2023.

[8] Valentina Lenarduzzi, Vili Nikkola, Nyyti Saarimäki, and Davide Taibi. Does code quality affect pull request acceptance? an empirical study. *J. Syst. Softw.*, 171:110806, 2021.

[9] NAIST-SE. Devgpt: A dataset of developer interactions with chatgpt for software development artifacts. `https://github.com/NAIST-SE/DevGPT`, 2023.

[10] Justine Winata Purwoko, Tegar Abdullah, Budiman Wijaya, Alexander Agung Santoso Gunawan, and Karen Etania Saputra. Analysis ChatGPT potential: Transforming software development with AI chat bots. In *2023 International Conference on Networking, Electrical Engineering, Computer Science, and Technology (IConNECT)*, pages 36–41. IEEE.

[11] Md Fazle Rabbi, Arifa Islam Champa, Minhaz F. Zibran, and Md Rakibul Islam. Ai writes, we analyze: The chatgpt python code saga. *Proceedings of the 21st International Conference on Mining Software Repositories*, 4:177–181, Apr 2024.

[12] Mohammad Masudur Rahman and Chanchal K. Roy. An insight into the pull requests of github. *CoRR*, abs/1807.01853, 2018.

[13] Mohammad Masudur Rahman and Chanchal K. Roy. An insight into the pull requests of github. *CoRR*, abs/1807.01853, 2018.

[14] Wahyu Rahmaniar. Chatgpt for software development: Opportunities and challenges. *IT Prof.*, 26(3):80–86, 2024.

[15] Partha Pratim Ray. Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems*, 3:121–154, 2023.

[16] Daricélio Moreira Soares, Manoel Limeira de Lima Júnior, Leonardo Murta, and Alexandre Plastino. Acceptance factors of pull requests in open-source projects. In *SAC*, pages 1541–1546. ACM, 2015.

[17] Christoph Treude, Ohad Barzilay, and Margaret-Anne D. Storey. How do programmers ask and answer questions on the web? In *ICSE*, pages 804–807. ACM, 2011.

[18] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C. Schmidt. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. *CoRR*, abs/2303.07839, 2023.

[19] Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long Han, and Yang Tang. A brief overview of chatgpt: The history, status quo and potential future development. *IEEE CAA J. Autom. Sinica*, 10(5):1122–1136, 2023.

[20] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. Devgpt: Studying developer-chatgpt conversations. *CoRR*, abs/2309.03914, 2023.

[21] Zhou Yang, Zhensu Sun, Terry Yue Zhuo, Premkumar T. Devanbu, and David Lo. Robustness, security, privacy, explainability, efficiency, and usability of large language models for code. *CoRR*, abs/2403.07506, 2024.

[22] Burak Yetistiren, Isik Özsoy, Miray Ayerdem, and Eray Tüzün. Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt. *CoRR*, abs/2304.10778, 2023.