Research Internship Report
Software Science

Radboud University

Internship report with some cool name

Author:
Elina Antonova
s1057069

Supervisor:
dr. Mairieli Wessel
`mairieli.wessel@ru.nl`

May 14, 2025

Abstract

Very cool abstract to my research

Contents

Chapter 1

Introduction

Large Language Models (LLMs) are statistical models that were trained on the large amount of data containing billions of parameters and use previous tokens to predict the future tokens [1]. LLM are used in different domains such as conversation agents, education, text summarisation, explainable AI, information retrieval and many more [2]. One of the most well-known models is Chat Generative Pre-trained Transformer (ChatGPT) [3], a chatbot developed by OpenAI and first released in November 2022. According to OpenAI CEO Sam Altman, ChatGPT reached 100 million weekly active users in November 2023[1].

ChatGPT drew so much attention for its detailed and expressive answers across many domains of knowledge, including software development. The chatbot was adopted by some software engineers as an alternative to using Google for solving software development related issues. Studies have investigated ChatGPT's efficacy when used in software development processes [4], its limitations [1], domains of application within issue-tracking systems and reliance on the ChatGPT-generated code [5] and the correctness of the generated code [6, 7]. Additionally, some studies focus on the human-bot interaction: the role of ChatGPT in the process of collaborative software architecting [8] or what is ChatGPT's primary use in issue-tracking systems [5].

In this study we focus on the domain of human-bot interaction and use developer prompts to extract the different types of collaboration that developers want ChatGPT to follow during their interaction. We expect to see requests for ChatGPT to be a pair programmer and work together on issue resolution, where developer provides a feedback on how the chatbot answer can be improved and vice-versa; or it will be asked to review the code and suggest improvements or find a bug in the program. There are many ways developers can collaborate with ChatGPT and we focus on extracting some

[1]https://www.theverge.com/2023/11/6/23948619/openai-chatgpt-devday-developer-conference-news

patterns from their prompts to recognise what type of collaboration does developer expect from ChatGPT. This leads to the research question of this study: **What role does ChatGPT take in user-chatbot interaction with software developers?**

To answer the research question we use various supervised and unsupervised text mining techniques: n-grams extraction, sequence mining and topic modelling. These methods help us with understanding the common themes and topics present in developers' ChatGPT prompts and the expectations they place on ChatGPT in these prompts. Sequence mining and topic modelling provide information extracted from the dataset that describes the data, while n-grams is used to provide the initial overview of what are the most common words/word combinations present in the data we are working with. We apply the mentioned techniques on DevGPT [9] dataset that consists of developer-ChatGPT conversations sourced from GitHub[2] commits, pull requests, code files, issues and discussion, and from HackerNews[3].

The answer to this research question provides valuable insights into the role that ChatGPT plays in developer-chatbot collaboration. Additionally, it shows what tasks developers trust chatbot like ChatGPT to perform and what expect from ChatGPT in that role. By outlining the collaboration patterns of developer-chatbot interaction, we provide the ground for new research to develop or enhance chatbots' ability to help the software developer in the way that is the most expected from the chatbot.

We firstly provide some background information and related research in Chapter 2.2. Then we describe in details what dataset consists of, what pre-processing steps are taken, how the data minins techniques are applied, provide some statistics, outliers and selected methods limitations in Chapter 3. Chapter 4 focuses on describing and analysing the results. Chapter 5 provides an overview of what was done and what results were achieved with this study, and provides the discussion of the project and possible future research.

[2]`https://github.com`

[3]`https://news.ycombinator.com`

Chapter 2

Background

2.1 ChatGPT and interaction with it

ChatGPT is a large-scale, pre-trained language model developed by OpenAI and designed to engage in natural language conversations, utilising Natural Language Processing (NLP) techniques that enable generation of the human-like text with context and coherence. Its performance in generating creative and novel text and ability to adapt to the context opened the possibilities for applications in various fields, such as creative writing, marketing, advertising, technical problem solving, etc [10].

ChatGPT model is a transformer-based neural network [11] crafted to accomplish natural language processing tasks. Initially the prototype was launched on 30 November 2022, which became available to public on 30 January 2023. The model is trained on vast corpus of textual data that was gathered from various sources, such as articles, books, reviews, online conversations and other human-generated data. This allowed model to capture natural language patterns, nuances, complexities allowing ChatGPT to perform well in non-trivial dialogues on various topics.

This research focuses on the interactions with ChatGPT. Interaction with ChatGPT consists of *conversation*, which is a collection of pairs of user *prompt* and ChatGPT's reply. The comversation starts with user creating the initial *prompt* which usually consists of the problem description, question, background information, step-by-step instruction for ChatGPT or any other information relevant to the user needs. ChatGPT uses the information provided in the prompt to generate the *response*. After user receives the response they can either stop the conversation or continue it with providing the new prompt for ChatGPT to process. Such ChatGPT conversations are used in this reseatch to answer the posed research questions with extensive user prompt analysis.

2.2 Related works

DevGPT [9] is a well-researched dataset that provides various insights into the developer-ChatGPT interactions and the usability of ChatGPT in the field related to software engineering.

The DevGPT dataset provides great opportunities to evaluate how ChatGPT handles questions and requests related to code generation and debugging, and how well the generated code corresponds to the set standards of code security, cleanliness, and efficiency. Several studies analyzed various aspects of the generated code: some studies focused on the security aspect and many other studies on the quality aspect of the generated code. The studies done by Siddiq, et al. [12] and Rabbi, et al. [13] revealed that the generated and modified code contained several CWE (Common Weakness Enumeration)[1] vulnerabilities, and issues captured by the Pylint 3.0.2[2] code quality analysis tool. Moreover, Zhang, et al. [14] showed that at least 35% of the 98 generated Kubernetes manifests contained at least one instance of a code smell. Other studies that researched the generated Python and Java code have discovered various code safety and quality issues by analyzing the generated code [15, 16, 17]. Moratis et al. [16] and Siddiq et al. [17] investigated the direct use of generated code in the source code base and concluded that ChatGPT generated code has to be reviewed and often modified if it is to comply with the code safety and quality standards.

Researchers are also exploring the structure, topics, and outcomes of interactions between developers and ChatGPT. Several studies using the DevGPT dataset have focused on analyzing the developers' intent and the subjects of their requests. Study conducted by Sagdic, et al. [18] have used BERTopic [19] topic modelling NLP technique to create topic clusters, which can be manually labeled and added to the topic's taxonomy. They have discovered seventeen topics that can be placed in seven categories: advanced programming; front- and backend development; DevOps integration and practices; streaming, media and localization; data management; and other topics. Similar approach was used by Mohamed, et al. [20], where they have used four different classifiers to categorize all conversations by their purposes based on the developer needs. Their research revealed 19 categories, out of which code generation and seeking general information accounted for almost third (31.15%) of all the conversations. Champa, et al. [21] have used *facebook/bart-large-mnli* model for zero-shot classification [22] to categorize the types of tasks developers present to ChatGPT, where they discovered that quality management of Python code and commit issue resolution tasks were the categories developers seeked the most assistance with. While all three studies share a common goal of categorizing developer interactions

[1]`https://cwe.mitre.org`

[2]`https://pypi.org/project/pylint/3.0.2/`

with ChatGPT, they differ in methodology and focus — ranging from topic modeling to classification based on intent — yet consistently highlight code generation and problem-solving as dominant use cases.

Recent studies have analyzed various aspects of integrating ChatGPT into multi-language software development and pull request workflows. Aguiar, et al. [23] found that over 75% of ChatGPT's code suggestions used a different language than the host repository, with the highest mismatch in CSS (91.16%) and the lowest in C# (60.18%) and CodeQL (58.08%). Chouchen, et al. [24] examined the topics of ChatGPT-related pull request conversations, identifying common themes such as code generation, refactoring, bug fixing, concept explanation, DevOps, testing, and recommendations, and noted that ChatGPT was more frequently used in large, time-consuming pull requests. Watanabe, et al. [25] and Hao, et al. [26] found that developers generally responded positively to ChatGPT's code review comments, with fewer than one-third of replies expressing disagreement. Additionally, Hao, et al. [27] observed that ChatGPT was used across various roles in GitHub pull requests and issues, including as an author, reviewer, and collaborator. Together, these studies highlight ChatGPT's growing presence and generally positive reception in collaborative software development environments. However, the responses should be critically checked as they still tend to contain code snippets written in non-native language.

Several recent studies have analyzed ChatGPT's effectiveness in addressing code generation and code refactoring tasks. Grewal, et al. [28] and Jin, et al. [29] investigated its use for code generation, concluding that while ChatGPT can produce functional code, developers often need to verify its correctness and adjust the structure to meet the functional requirements. Code refactoring, on the other hand, is a more open-ended task, where ChatGPT tends to focus on improving readability, maintainability, usability, and performance, as shown by Chavan, et al. [30] and AlOmar, et al. [31]. Their research revealed that developers typically adopt a straightforward strategy — copying the code and directly requesting a refactor — which resolves the issue in an average of 2.7 to 4.6 prompts. Prompt quality is also a critical factor in successful interactions with ChatGPT [32, 33]. Mondal, et al. [32] found that vague or underspecified prompts, especially those requesting additional features without clear specifications, often lead to longer and less efficient conversations. Altogether, these studies suggest that while ChatGPT is a valuable tool for code-related tasks, its effectiveness heavily depends on prompt clarity and developer oversight.

Chapter 3

Methodology

The purpose of investigating how developers use ChatGPT for their projects is to gain the necessary knowledge of the roles chatbot plays in developer-bot collaboration in order to enhance its ability to help developers with their requests. In order to extract the knowledge we firstly cleaned the data, investigated what it consists of and applied topic modelling and sequence mining in order to extract the patterns from developer-chatbot collaboration.

3.1 Data

To answer the research question we have used DevGPT dataset [9], which consists of the ChatGPT conversations that were shared on GitHub and Hacker News between late May 2023 and the 12th of October 2023. The dataset consists of 9 snapshots (latest: 12th of October, 2023), and each snapshot contains the links collected from GitHub commits, issues, discussion, pull requests, code files, and Hacker News (HN) articles. In this research we use the data from all of the available snapshots. However, to keep the data clean, we remove all the duplicate data based on the shared ChatGPT conversation links. Table 3.1 shows the number of conversations that is left after all the duplicates were removed from the dataset (*DR* row).

discuss that DevGPT dataset was extended with more data

	Commits	Issues	Discussions	Pull req.	Code files	HN
DR	670	516	59	268	2010	322
NE	659	433	51	227	1616	290

Table 3.1: Number of conversations after all the duplicates (DR) and non-English conversations (NE) were removed from the data.

3.2 Data cleaning

At the initial investigation of the data, we discovered that the data contains some information that is not needed to answer the research question of this paper (further referred to as "noise"). Firstly, we only focus on conversations in English, while the dataset contains other languages used for communication with ChatGPT. Secondly, in addition to natural language, the prompts contain program code as a part of the prompt text. Both these factors make the data noisy and require to be dealt with before the data exploration and RQ answering is done.

3.2.1 Language detection

For the language detection purposes we have used existing libraries: *langdetect*[1] and *pycld2 (Compact Langauge Detect 2)*[2]. These two libraries are used to ensure higher accuracy at language detection: if one of the libraries is unable to detect the language, then the other library results influence the decision whether to keep the prompt (if it is detected as English text) or discard.

For language identification process the prompt is divided into lines and the language is identified for each line. If the line consists of the programming code only, then the line is marked as program (details in Section 3.2.2) and the language identification is skipped for this line. This is done in order to avoid prompts in non-English language containing a lot of code lines, which have a lot of English words/keywords, to be identified as an English prompts. The result or language check is a tuple, where the first value is the language identification result of *langdetect* and the second value is *pycld2* result. If the mode language of all the prompt lines is either ("*en*","*en*"), ("*en*","*un*") or ("*un*","*en*"), then the language of the prompt is considered to be English. Any other language detection result leads to the prompt being discarded.

3.2.2 Program lines detection and its accuracy

Initially we tried to implement the programming language detection using Guessland python library[3], but its accuracy and recall were very low for the dataset that we are using. So it was decided to implement our own code for detecting programming lines in order to remove them before the data analysis is conducted.

In order to detect the code parts in the prompts, we split all the prompts into lines at line break ($\backslash n$) and used the script to detect the probability

[1] `https://pypi.org/project/langdetect/`

[2] `https://pypi.org/project/pycld2/`

[3] `https://pypi.org/project/guesslang/`

of the line being a part of code or not. This is done with the assumption that the code blocks are most likely to be present on the separate lines from the natural language sentences and that the code is likely to be spread over multiple lines of the prompt. Thus, the code identification needs to be performed on each line of the code block. If the line contains a code snippet among natural language text, but the natural language part is longer than the code snippet, then the line is identified as a natural language line. In order to detect if the line is programming language or not, we created the function that assigns the line structures (tokens or sentences) with the number that represents how likely the structure is natural language: 0 - programming language, 0.5 - can be either, 1 - natural language. The list below shows what kind of structures are considered in this script:

- Line starts with `//` or `/*` (C-lamguages, Java, etc.), `--` (Haskell), `#` or `"""` (Python), `<!--` (HTML) or ends with `*/`, `"""` or `-->` indicates that the line is a part of comment within the program (likelyhood: 0);
- Line starts opening HTML tag (e.g. `<p>`, `<h1>`, etc.) and ends with closing HTML tag (e.g. `</p>`, `</h1>`, etc.) indicates that the line is a part of HTML code (likelyhood: 0);
- Structure is a keyword in Java, C (C++, C#), HTML, CSS, Python, Command Prompt or Linux command (likelyhood: 0.5);
- Structure is upper case (also covers SQL keywords) (likelyhood: 0.5);
- Structure is written in camel or pascal case (likelyhood: 0);
- Structure only consists of letters (likelyhood: 1);
- Structure consists of letters and comma, period, colon, question or exclamation mark (likelyhood: 0.5);
- Structure is a word surrounded with square brackets (likelyhood: 0);
- Structure has no letters or is empty string (likelyhood: 0);
- Structure starts with ::, :: −, :, #, ., < \\, < followed by a word (likelyhood: 0);
- Structure is a word surrounded with less-that and greater-than signs (likelyhood: 0);
- Structure is a combination of two words with a dot in between (likelyhood: 0);
- Structure has a form of function or function call: $function\_name(args)$, where $args$ is a list of 0 or more arguments separated by the comma (likelyhood: 0);

The likelihood of line having a natural language text in it is calculated by summing all the likelyhoods together and dividing by the number of structures in the line. If the result is abive 0.5, the line is considered to contain the natural language text in it, if the likelyhood is lower, then the line is most likely the code line.

The usability of the program code detection script is tested using 5 randomly selected conversations and taking up to 20 randomly selected lines out of these 5 conversations. The selected lines are sent to the script and the script results were saved into the file, where the line content was manually evaluated to be the program code or not. The results of the script performance were satisfactory for all the conversation sources:

- GitHub commits: 5 false-positives and 2 false-negative (accuracy: 0.88, F1-score: 0.9;
- GitHub issues (accuracy: 1, F1-score: 1);
- GitHub discussions: 1 false-positives and 0 false-negative (accuracy: 0.99, F1-score: 0.98);
- GitHub pull requests: 1 false-positives and 1 false-negative (accuracy: 0.95, F1-score: 0.97);
- GitHub code files: 2 false-positives and 0 false-negative (accuracy: 0.97, F1-score: 0.98);
- Hacker News (accuracy: 1, F1-score: 1);

The false-positive and false-negative results are printed out to inspect what mistakes are made by the written script. Most common mistakes are HTML and SQL code lines that are marked as "not a code", due to the high number of natural language structures they contain. For example, HTML paragraphs `<p>` contain whole sentence(s) written in natural language, which influence the final result of the script. Sentences containing high number of punctuation marks, on the other hand, are often marked as the program, even when they are not. For example, lines *"perfect!"* and *"- Prefer *async/await* over promises!"* were predicted to be a program, however, it can also be a part of text. Despite that, the results of the script are on average above 85% accuracy and F1 score, which we consider to be a good result for this research.

3.3 Data exploration

As a part of data exploration, we focus on gathering some statistics about the data set: how long the conversations are, how many symbols or words is each prompt on average, the most common n-grams and topics that can be extracted from the prompts.

Before going into discussion about the statistics of the data collected, it is important to explain some vocabulary used in this section. Users write *prompts* to ChatGPT, which uses them to formulate the answer. The collection of such prompt-answer pairs that happens in the same conversation thread is called *conversation*. In this dataset we only focus on the prompts written by the user and discard the answers generated by ChatGPT in all the conversations.

3.3.1 Statistics and outliers

This section covers some statistics regarding the length of the conversations and prompts and the outliers encountered during data exploration process.

Number of prompts in conversations

The number of prompts in the conversations varied significantly for the conversations obtained from the different sources. The shortest conversations were found at GitHub discussion dataset, while GitHub code files dataset contained longest discussions among all the datasets. Table 3.2 shows the statistics of the conversation lengths for every dataset obtained from different sources. The median conversation length values for all the conversation sources are similar to each other, staying between 1 and 7 prompts per conversations. But when it comes to the mean and standard deviation, the difference becomes more significant. GitHub commits, issues and code files related conversations are considerably longer than the conversations from the other sources. This can be concluded from the higher mean of 23.1, 21.7 and 22.6 prompts and very high standard deviations of 85.4, 59.4 and 100.5 prompts for GitHub commits, issues and code files, respectively. These high numbers are mainly influences by the outliers present in these datasets that are discussed later in this section. The other datasets had lower mean and standard deviation values, where mean and standard deviation seem to represent the data quite well.

Source	Median	Mean	Standard deviation
GH Commits	2	23.1	85.4
GH Issues	3	21.7	59.4
GH Discussions	3	6.7	10.4
GH Pull requests	1	4.9	19.9
GH code files	5	22.6	100.5
HackerNews	7	17.6	40

Table 3.2: Median, mean and standard deviation of the conversation lengths from different sources.

Figure 3.1 contains plots where the line shows how many conversations are there per number of the prompt. Axis x represents the prompt number, where minimum value is 1 and maximum value is the number of prompts in the longest conversation of this dataset. Axis y shows how many conversations have x or more prompts at any x-value. The figure shows that all the conversations sources from GitHub have some conversations that are significantly longer than the rest of the conversations in the dataset. A good example is one conversation from pull requests, that is about 133 prompts longer than the next longest conversation in this dataset. The conversations that seemed to be outliers in each of the datasets were saved to the file for the further investigation of their content. In each dataset there were conversations, which prompt numbers are much greater that the prompt numbers of the next longest conversations. In order to capture them, we have selected the following cut-off points for the conversation lengths, and saved the ones that were longer or equal to these points:

- GitHub commits: conversation length of 30 or more;
- GitHub issues: conversation length of 30 or more;
- GitHub discussions: conversation length of 15 or more;
- GitHub pull request: conversation length of 30 or more;
- GitHub code files: conversation length of 100 or more;
- Hacker News: conversation length of 50 or more;

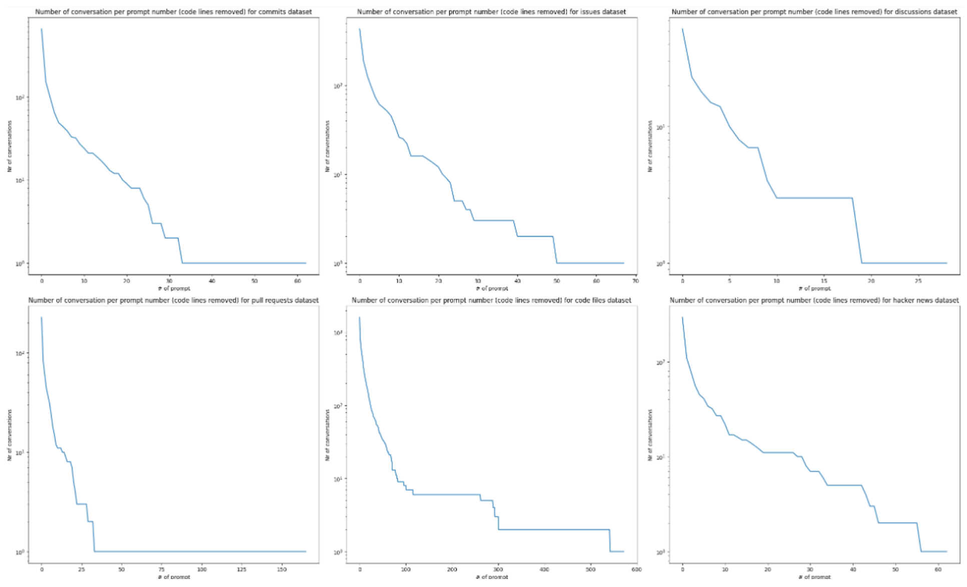

Figure 3.1: Number of conversations per prompt number for all conversation sources.

Outliers and their content

The conversations that are too long are saved to the file and investigated manually for the reasons why they contain that many prompts. Doing that we managed to find the following reasons on why they were that long:

- Conversation contains several prompts of text that did not fit into one prompt due to the set character limit: users were copying the whole code files, articles or books.
- Users were asking for code snippets, and the code provided from ChatGPT either did not correspond to user expectations and the request had to be modified several times by the user, or the code returned the errors, that users were not able to solve and had to ask for help from ChatGPT.
- Users used ChatGPT for programming: they provided with the information what they want it to program, asking ChatGPT to write the code, while testing it on their machine and providing with the feedback on the errors received or what needs to be changed.
- Users use ChatGPT as their mentor: they post the code they wrote asking for explanation or improvements suggestions. Occasionally posting errors it produced and asking for help or explanation.

- Users use ChatGPT for conversations that are not related to the programming issues: talking about ChatGPT opinions on different topics, asking for advice or asking more general questions;

Average symbol/word count per prompt

The number of symbols and words per prompts varied significantly depending on the source of the conversation. However, the graphs for the number of symbols and the number of words per each source are comparable, where the peaks and troughs on the graphs are present at the same locations. Thus, we will look at both the number of symbols and the number of words together to investigate the outliers that have too high symbol/word count.

Figure 3.2 and 3.3 show the average number of symbols and words per current prompt number, respectively. Axis x represents the prompt number, where minimum value is 1 and maximum value is the number of prompts in the longest conversation of this dataset. Axis y shows the average count of symbols/words that prompt x has. One important factor that influences the way graphs look like is the length of conversations. The graphs show average count of symbols/words for the prompt number: some conversations are shorter than others, meaning that longer conversations have more influences on the average symbol/word count for the higher prompt counts, since the average is calculated on the lower number of prompts. A good example of it is GitHub discussion dataset, which has only one conversation longer than 19 prompts, meaning that the average symbol/word count for prompts 20 to 29 is taken from one conversation, which results in peaks on the graphs. GitHub code files dataset shows different behaviour: longest conversations have lower symbol/word counts. However, in all datasets first prompts are usually used to introduce the problem ChatGPT needs to solve, which requires a more thorough description of the problem; thus, resulting in a higher symbol/word count.

To investigate the outliers, we have selected the cut-off points for each dataset separately. The cut-off point is selected to be 2/3 of the maximum average. Conversations containing prompts longer than the cut-off points were saved for the further investigation. GitHub commits and HackerNews datasets have smooth graphs with little fluctuation, thus, they are not included in outlier investigation process. For all the other four datasets we have selected the following cut-off points:

- GitHub issues: prompt length in symbols of 400 or more;
- GitHub discussions: prompt length in symbols of 800 or more;
- GitHub pull request: prompt length in symbols of 800 or more;
- GitHub code files: prompt length in symbols of 850 or more;

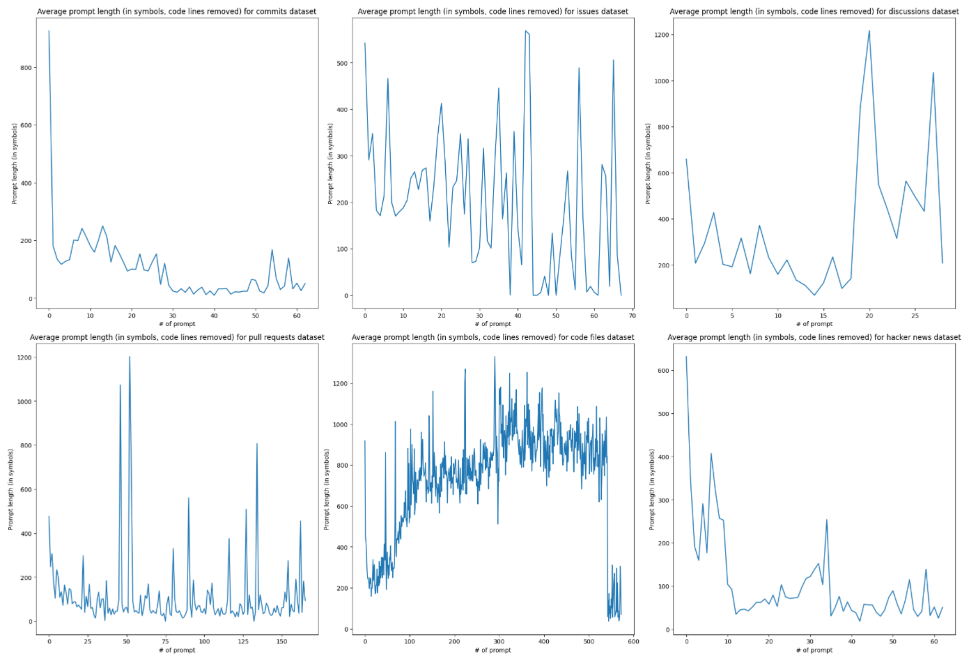

Figure 3.2: Average symbol count per prompt number for all conversation sources after code lines were removed.

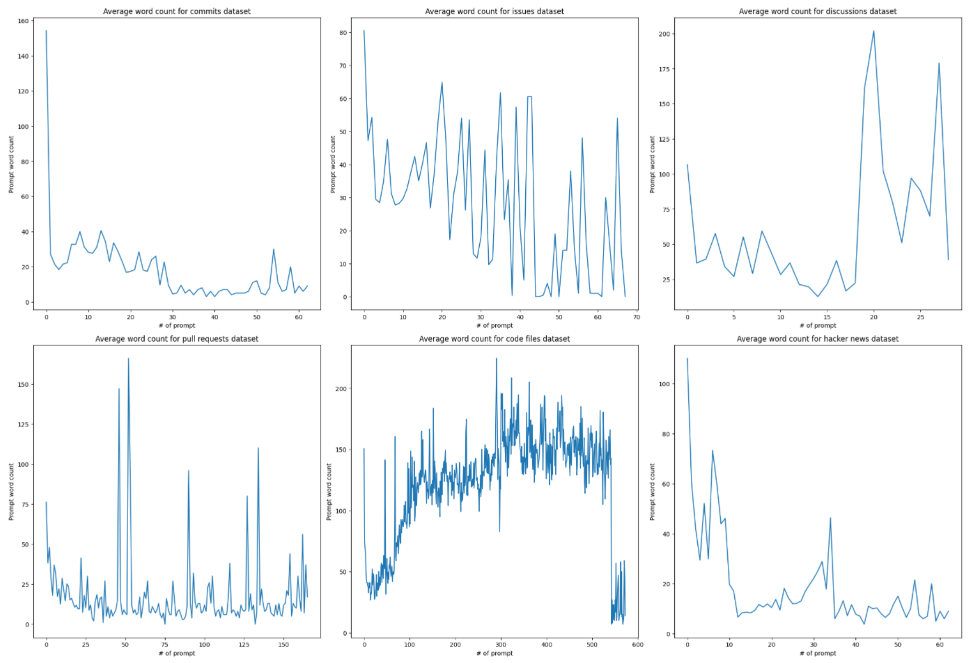

Figure 3.3: Average word count per prompt number for all conversation sources after code lines were removed.

Outliers and their content

The conversations containing very long prompts are saved to the file, and their content is investigated manually to identify the reasons that explain their length. We have managed to identify the following content categories in long prompts:

- Prompt contains long text that was pasted by the user from the external source (scientific article, GitHub issue description, code documentation, etc.), and user wants to get the analysis or explanation of the text or its part, or wants to get the rewritten version to match the target audience.
- Prompt contains error message, code documentation/comments and code parts (that were not detected by program line detection function) that user wants to fix, debug, get feedback on.
- Prompt contains a thorough description of the problem the user is facing, which can contain:
  - A very detailed problem description, when the user knows exactly what they want to receive from ChatGPT;
  - A problem description in a poetic way (usually asking ChatGPT to play a certain role: e.g. teacher, mentor, area expert);
  - User provides relevant for the problem background information to ChatGPT;
  - Multiple questions regarding the problem or provided solution;
  - What solutions user found, which did not work;
  - Long URL links;
  - The list of requirements or instructions;
  - Example of the output, functionality, or code user wants to receive.

3.4 Data preparation

In order to perform further analysis, we have pre-processed the remaining data using lemmatisation and tokenisation processes.

During lemmatisation process we split sentences into a list of structures: words, punctuation, digits or alphanumeric values; and lemmatise them using *lemmatize_sentence* function from **pywsd** package [34]. The function tries to convert a surface word into lemma, and if lemmatize word is not in wordnet then try and convert surface word into its stem.

Once all the sentences are lemmatised, they are cleaned from the non-relevant data. We remove the following tokens:

- Links: token starts with *"http"*;
- Stop words: the selection of *stop_words* from **nltk** package[4] [35];
- Punctuation;
- Word contains digits.

Lemmatised and cleaned lists of tokens are then further passed on to the functions that conduct further analysis on the content of the cleaned sentences.

3.5 N-grams

N-grams is a sequence of n consecutive characters, words or other textual structures [36], where n is the number of such structures in a pair. The set of n-grams is obtained by shifting the box of **n** structures through the data and retrieving the values. The structures order is kept when the n-grams are being extracted. In this research we focus on the words, so the sentence *"That fox is chasing a rabbit"* can be split into the following structures: $that, fox, is, chasing, arabbit$. We can create the following n-grams from the mentioned set of words:

- Bi-grams: $(that, fox)$, (fox, is), $(is, chasing)$, $(chasing, a)$, $(a, rabbit)$;
- Tri-grams: $(that, fox, is)$, $(fox, is, chasing)$, $(is, chasing, a)$, $(chasing, a, rabbit)$;
- Quadri-grams: $(that, fox, is, chasing)$, $(fox, is, chasing, a)$, $(is, chasing, a, rabbit)$;

Before the data is used for n-grams extraction, the data is cleaned from stop words, as described in Section 3.4. In the example above it would mean that words *"that"*, *"is"* and *"a"* will be removed in the pre-processing step. All the other context carrying words are used to create n-grams for the data. N-grams are created using *ngrams()* function from **nltk** package, where n is set to the desirable value. The extracted n-grams are used to create n-grams frequency distribution: all the same n-grams are grouped together and their frequency is calculated. Such frequency distribution allows us to order all the extracted n-grams in descending order based on their frequency and focus our analysis on the most frequent n-grams.

For easier visualisation, we use WordCloud package [37] that generates word clouds where the most common n-grams use bigger font size, while less common ones use smaller font sizes that is dependent on their frequency.

[4]Stop words list

3.6 Topic modelling

Topic models are generative models that provide a probabilistic framework [38]. These models are used for large electronic text collections to organise, understand, search and summarise their content.

The topics are the relations that link words in a vocabulary and their occurrence in documents. Each document is viewed as a mixture of topics. Topic models discover different themes present in the text collections and annotate the documents according to the themes. The document coverage distribution of topics is then generated to provide an overview of topics found in this document collection.

In this study we use Latent Dirichlet Allocation (LDA) topic modelling – one of the most popular text modelling techniques. LDA is a probabilistic generative model that allows the observations to be described by unobserved data that explains why some parts of the data are similar to each other [39]. In LDA, documents consist of multiple topics and each topic is a distribution over a fixed vocabulary. For example, if we have the following vocabulary of {*pan, cook, football, basketball*}, then the kitchen topic will assigne high probabilities to the words *pan* and *cook*, while *football* and *basketball* will have very low probabilities; however, the sport topic will have the opposite probabilities for all the words.

In the implementation we use the data cleaned in the pre-processing step as described in Section 3.4. We start with creating the vacabulary: this is done using *CountVectorizer* class to convert a collection of text documents to a matrix of token counts [40]. The parameters of *min_df* and **max_df** are set to 0 and 1 respectively; *ngram_range* is set to tuple of (1,4), making vocabulary to consist of n-grams of the length 1 to 4 words; and the stop words are the stop words from **nltk** package. For topic modelling we use *LatentDirichletAllocation* class from **sklearn** package [41], where we set number of topics to 10, iteration count to 6 and random state to 42. We calculate perplexity and coherence score of the topic model and visualise them along with WordCloud visualisation of the topics.

Code: remove split, use fit_transform

3.7 Sequence mining

add

3.8 Limitations

There exist limitations of the dataset and methods selected for this study that influence the results of the research. These limitation are addressed in this section.

Dataset used for this study provides a good overview of how developers use ChatGPT for solving their daily problems. However, the sources that were used to extract this knowledge are not completer. The conversation data was collected from GitHub (commits, code files, discussions, pull requests and issues) and HackerNews, and it covers the conversations that happened between the release date of ChatGPT and 12th of October 2023. Thus, the data is not complete and misses other sources that could contain more knowledge regarding the use of ChatGPT by developers. Additionally, since the data was last sourced over a year ago, the conversations that happened after the mentioned date and conversations with ChatGPT that uses newer version GPT-4[5] are not included in this dataset. Moreover, some conversation links that were used to scrape the data are not working anymore, thus, the validity of these conversations cannot be checked.

Additionally, the language detection tool and the data cleaning method used are not fully accurate and miss prompts or prompt lines that contain foreign language and code. It was discovered, that the prompts often contain pasted text (articles, documentation, code comments) and error messages, which are hard to detect using the selected tool set, since they contain sentences in natural English language, but do not contain the information needed for answering the research question. Thus, the code snippets, foreign language prompts and copied texts add noise to the data that the tools are not able to detect.

[5]`https://openai.com/index/gpt-4/`

Chapter 4

Results

In this chapter we provide insights into the knowledge we managed to extract during the analysis of developers prompts in developer-ChatGPT interaction. This chapter focuses on the results extracted from statistical analysis, N-grams, topic modelling and sequence mining of the prompts.

4.1 Statistics and outliers

4.2 N-grams

4.3 Topic modelling

4.4 Sequence mining

Chapter 5

Discussion and conclusion

Bibliography

[1] A. Azaria, “ChatGPT Usage and Limitations.” working paper or preprint, Dec. 2022.

[2] R. Dale, “Gpt-3: What’s it good for?,” *Natural Language Engineering*, vol. 27, no. 1, p. 113–118, 2021.

[3] L. Floridi and M. Chiriatti, “Gpt-3: Its nature, scope, limits, and consequences,” *Minds and Machines*, vol. 30, p. 681–694, Nov. 2020.

[4] A. Hörnemalm, *ChatGPT as a Software Development Tool: The Future of Development*. PhD thesis, 2023.

[5] J. K. Das, S. Mondal, and C. K. Roy, “Investigating the utility of chatgpt in the issue tracking system: An exploratory study,” 2024.

[6] J. Liu, C. S. Xia, Y. Wang, and L. ZHANG, “Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation,” in *Advances in Neural Information Processing Systems* (A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, eds.), vol. 36, pp. 21558–21572, Curran Associates, Inc., 2023.

[7] Q. Zhang, T. Zhang, J. Zhai, C. Fang, B. Yu, W. Sun, and Z. Chen, “A critical review of large language model on software engineering: An example from chatgpt and automated program repair,” 2023.

[8] A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, M. S. Aktar, and T. Mikkonen, “Towards human-bot collaborative software architecting with chatgpt,” in *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, EASE ’23, (New York, NY, USA), p. 279–285, Association for Computing Machinery, 2023.

[9] T. Xiao, C. Treude, H. Hata, and K. Matsumoto, “Devgpt: Studying developer-chatgpt conversations,” Nov. 2023.

[10] K. I. Roumeliotis and N. D. Tselikas, “Chatgpt and open-ai models: A preliminary review,” *Future Internet*, vol. 15, no. 6, 2023.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[12] M. L. Siddiq and J. C. Santos, "Generate and pray: Using sallms to evaluate the security of llm generated code," *arXiv preprint arXiv:2311.00889*, 2023.

[13] M. F. Rabbi, A. I. Champa, M. F. Zibran, and M. R. Islam, "Ai writes, we analyze: The chatgpt python code saga," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 177–181, Association for Computing Machinery, 2024.

[14] Y. Zhang, R. Meredith, W. Reeves, J. Coriolano, M. A. Babar, and A. Rahman, "Does generative ai generate smells related to container orchestration?: An exploratory study with kubernetes manifests," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 192–196, Association for Computing Machinery, 2024.

[15] A. Clark, D. Igbokwe, S. Ross, and M. F. Zibran, "A quantitative analysis of quality and consistency in ai-generated code," in *2024 7th International Conference on Software and System Engineering (ICoSSE)*, pp. 37–41, 2024.

[16] K. Moratis, T. Diamantopoulos, D.-N. Nastos, and A. Symeonidis, "Write me this code: An analysis of chatgpt quality for producing source code," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 147–151, Association for Computing Machinery, 2024.

[17] M. L. Siddiq, L. Roney, J. Zhang, and J. C. D. S. Santos, "Quality assessment of chatgpt generated code and their use by developers," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 152–156, Association for Computing Machinery, 2024.

[18] E. Sagdic, A. Bayram, and M. R. Islam, "On the taxonomy of developers' discussion topics with chatgpt," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 197–201, Association for Computing Machinery, 2024.

[19] M. Grootendorst, "Bertopic: Neural topic modeling with a class-based tf-idf procedure," 2022.

[20] S. Mohamed, A. Parvin, and E. Parra, "Chatting with ai: Deciphering developer conversations with chatgpt," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 187–191, Association for Computing Machinery, 2024.

[21] A. I. Champa, M. F. Rabbi, C. Nachuma, and M. F. Zibran, "Chatgpt in action: Analyzing its use in software development," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 182–186, Association for Computing Machinery, 2024.

[22] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," 2019.

[23] L. Aguiar, M. Paixao, R. Carmo, E. Soares, A. Leal, M. Freitas, and E. Gama, "Multi-language software development in the llm era: Insights from practitioners' conversations with chatgpt," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '24, (New York, NY, USA), p. 489–495, Association for Computing Machinery, 2024.

[24] M. Chouchen, N. Bessghaier, M. Begoug, A. Ouni, E. Alomar, and M. W. Mkaouer, "How do software developers use chatgpt? an exploratory study on github pull requests," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 212–216, Association for Computing Machinery, 2024.

[25] M. Watanabe, Y. Kashiwa, B. Lin, T. Hirao, K. Yamaguchi, and H. Iida, "On the use of chatgpt for code review: Do developers like reviews by chatgpt?," EASE '24, (New York, NY, USA), p. 375–380, Association for Computing Machinery, 2024.

[26] H. Hao and Y. Tian, "Engaging with ai: An exploratory study on developers' sharing and reactions to chatgpt in github pull requests," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering Workshops*, ASEW '24, (New York, NY, USA), p. 156–160, Association for Computing Machinery, 2024.

[27] H. Hao, K. A. Hasan, H. Qin, M. Macedo, Y. Tian, S. H. H. Ding, and A. E. Hassan, "An empirical study on developers' shared conversations

with chatgpt in github pull requests and issues," *Empirical Software Engineering*, vol. 29, p. 150, Sep 2024.

[28] B. Grewal, W. Lu, S. Nadi, and C.-P. Bezemer, "Analyzing developer use of chatgpt generated code in open source github projects," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 157–161, Association for Computing Machinery, 2024.

[29] K. Jin, C.-Y. Wang, H. V. Pham, and H. Hemmati, "Can chatgpt support developers? an empirical evaluation of large language models for code generation," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 167–171, Association for Computing Machinery, 2024.

[30] O. S. Chavan, D. D. Hinge, S. S. Deo, Y. O. Wang, and M. W. Mkaouer, "Analyzing developer-chatgpt conversations for software refactoring: An exploratory study," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 207–211, Association for Computing Machinery, 2024.

[31] E. A. AlOmar, A. Venkatakrishnan, M. W. Mkaouer, C. Newman, and A. Ouni, "How to refactor this code? an exploratory study on developer-chatgpt refactoring conversations," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 202–206, Association for Computing Machinery, 2024.

[32] S. Mondal, S. D. Bappon, and C. K. Roy, "Enhancing user interaction in chatgpt: Characterizing and consolidating multiple prompts for issue resolution," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 222–226, Association for Computing Machinery, 2024.

[33] L. Wu, Y. Zhao, X. Hou, T. Liu, and H. Wang, "Chatgpt chats decoded: Uncovering prompt patterns for superior solutions in software development lifecycle," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 142–146, Association for Computing Machinery, 2024.

[34] L. Tan, "Pywsd: Python implementations of word sense disambiguation (wsd) technologies [software]." https://github.com/alvations/pywsd, 2014.

[35] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[36] Z. Elberrichi and B. Aljohar, "N-grams in texts categorization," *Scientific Journal of King Faisal University (Basic and Applied Sciences)*, vol. 8, no. 2, pp. 25–39, 2007.

[37] A. C. Mueller, "Wordcloud," 2023.

[38] B. Grün and K. Hornik, "topicmodels: An r package for fitting topic models," *Journal of Statistical Software*, vol. 40, no. 13, p. 1–30, 2011.

[39] Z. Tong and H. Zhang, "A text mining research based on lda topic modelling," in *International conference on computer science, engineering and information technology*, pp. 201–210, 2016.

[40] "CountVectorizer — scikit-learn.org." `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html`. [Accessed 23-09-2024].

[41] "LatentDirichletAllocation — scikit-learn.org." `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html`. [Accessed 23-09-2024].

Glossary

ChatGPT	Chat Generative Pre-trained Transformer. 2–4, 7, 8, 11, 13, 14, 16, 18–20
LDA	Latent Dirichlet Allocation. 18
LLM	Large Language Model. 2
NLP	Natural Language Processing. 4, 5

Reflection on research process

- Reflection on the research results
- Reflection on the process
- Problems encountered
- Takeaways

Appendix A

Appendix