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

June 12, 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 4. Chapter 5 focuses on describing and analysing the results. Chapter 6 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

Data

To answer the research question, we used the DevGPT dataset [9], which contains ChatGPT conversations shared on GitHub and Hacker News between late May 2023 and October 12, 2023. The dataset includes nine snapshots (latest: October 12, 2023), each comprising links to ChatGPT conversations found in GitHub commits, issues, discussions, pull requests, code files, and Hacker News articles. Considering the origins the conversations were sourced from, it is safe to assume that the prompts for this conversations were written by software developers.

In this study, we use data from all available snapshots. To ensure data quality, we removed duplicate entries based on shared ChatGPT conversation links. Additionally, we extended the DevGPT dataset with a new dataset collected in March 202. This extension contains developer-chatbot conversations shared in the same sources but posted after the latest DevGPT snapshot. The structure of the extended dataset mirrors that of DevGPT to maintain consistency. The dataset was provided for this research by author's colleagues from the same research group.

Table 3.1 presents the number of unique conversations sourced from each data source, along with the remaining number of conversations after removing duplicates and non-English conversations.

	Commits	Issues	Discussions	Pull req.	Code files	Repositories	Hacker News	Total
DevGPT DR	670	516	59	268	2010	0	322	3845
Extension DR	881	1134	65	775	9536	10	805	13206
Combined DR	896	1235	113	813	9959	10	858	13884
Combined NE	862	998	89	682	8057	8	773	11469

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

The total number of conversations shared across all the sources after duplicates and non-English conversations were removed is 11469. However,

after removing duplicate conversations from the entire combined dataset, it leaves us with 10279 conversations, which means that 1190 developer-ChatGPT conversations are shared in multiple locations.

3.1 Data cleaning

During our initial analysis of the dataset, we identified the presence of information irrelevant to the research question — referred to as "noise". First, many prompts contain embedded code alongside natural language. A good example of this is the following prompt from DevGPT dataset sourced from GitHub Issues:

So you want me to hard code the password in this section of src/pages/api/getDeviceInfo.js?

$$
\begin{aligned}
&const\ pool = new\ Pool(\{\\
&\quad host : process.env.PGHOST\_2,\\
&\quad port : process.env.PGPORT\_2,\\
&\quad database : process.env.PGDATABASE\_2,\\
&\quad user : process.env.PGUSER\_2,\\
&\});
\end{aligned}
$$

Second, while we focus exclusively on English-language conversations, the dataset includes interactions in multiple languages. For example, the following prompts was sourced from conversation shared in GitHub commits: *mach mir noch so eine aufgabe zu dem selben muster, damit ich das thema besser verstehen kann.*

Both of the mentioned factors introduce noise that must be removed before proceeding with data exploration and answering the research questions.

3.1.1 Code detection and its accuracy

Initially we attempted to detect programming language content using the Guesslang Python library[1], but its accuracy and F1-score were too low for our dataset. As a result, we developed a custom method for identifying and filtering out programming lines.

To identify code segments within prompts, we first split each prompt into individual lines using the line break character ($\backslash n$). We then applied a custom script to estimate the likelihood that each line contains code. This approach is based on the assumption that code blocks typically appear on separate lines from natural language text and often span multiple lines within a prompt. As a result, line-by-line analysis is necessary to accurately detect code.

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

To identify whether a line is written in a programming language or natural language, we implemented a function that evaluates the line's structures — whether they are tokens or full sentences — and assigns them a probability score: `0` indicates a programming language structure, `0.5` indicates ambiguity (could be either), and `1` indicates natural language. The list below shows what kind of structures are considered in this script:

- Line is empty (likelihood: 0);
- Line starts a multiline comment block with symbols like `/*`, `"""`, `'''`, or `<!--`, and does not end on the same line — the whole block is marked as comment (likelihood: 0);
- Line is within a multiline comment block and ends with symbols like `*/`, `"""`, `'''`, or `-->` — marked as comment (likelihood: 0);
- Line matches single-line comment patterns such as `//`, `#`, `--`, etc. — marked as comment (likelihood: 0);
- Line starts with an opening HTML tag (e.g. `<div>`) and does not end with a closing tag — line and subsequent lines until closing tag are marked as HTML (likelihood: 0);
- Line contains a complete HTML block (e.g. `<p>Text</p>`) — marked as HTML (likelihood: 0);
- Line is tokenized using `shlex.split` if it contains quotes, otherwise by whitespace. Each token (word) is assigned a probability score based on specific heuristics:
  - Word contains space and resembles a sentence — averaged likelihood from subword analysis;
  - Word starts with tab (likelihood: 0);
  - Word is in a list of known programming keywords (likelihood: 0.5);
  - Word is a camelCase or PascalCase identifier (likelihood: 0);
  - Word is in UPPERCASE (e.g. SQL keywords; likelihood: 0.5);
  - Word is purely alphabetical (likelihood: 1);
  - Word is a comment-like or inline programming pattern (e.g. `::`, `#`, `<\`, `<|-`; likelihood: 0);
  - Word matches function calls, method chains, array indexing, or regular programming syntax (e.g. `func(x)`, `obj.attr`, `arr[i]`; likelihood: 0);
  - Word ends with punctuation (e.g. `word,`, `word.`, `word:`; likelihood: 0.5);

- Word ends with semicolon often used in programming syntax (likelihood: 0);
- Word lacks alphabetic characters (likelihood: 0);
- All other cases (likelihood: 1).

To determine whether a line is written in natural language or code, the script calculates the average likelihood score by summing the individual scores of all structures (tokens or sentences) in the line and dividing by the total number of structures. If the resulting average is above 0.5, the line is considered to contain natural language; otherwise, it is classified as a code line.

The effectiveness of the code detection script was evaluated using ten randomly selected conversations, from which up to 20 lines were randomly sampled. These lines were processed by the script, and the results were saved to a file. Each line was then manually labeled as either code or non-code for comparison. The script demonstrated satisfactory performance across all tested conversation sources. Table 3.2 shows how well the code detection script works.

	Accuracy	F-score
GitHub commits	0.96	0.98
GitHub issues	0.92	0.95
GitHub discussions	0.92	0.93
GitHub pull requests	0.96	0.97
GitHub code files	0.95	0.96
GitHub repositories	0.91	0.94
Hacker News	0.92	0.94
Total	**0.92**	**0.95**

Table 3.2: Accuracy and F-score of the code detection script per data source category.

False-positives and false-negatives were examined to better understand the types of errors made by the code detection script. The most common false-negatives occurred with error messages or log outputs, which often contain large amounts of natural language, making them difficult to distinguish from regular text. On the contrary, sentences with a high number of punctuation marks were frequently misclassified as code, even when they were not. For instance, lines such as *"Done."* and *"- Prefer async/await over promises!"* were incorrectly identified as code, although they could plausibly be part of natural language text.

This evaluation method also has limitations, particularly in handling multi-line HTML or comment blocks. Because lines for testing are selected randomly, it is possible to include a line that starts a multi-line HTML tag or comment block without capturing the corresponding closing line. As a result, following lines may be incorrectly labelled as part of HTML or a comment. Additionally, some lines contain both code and natural language, making results of the code detection depend on their proportion. Examples of such lines are questions that contain short code snippets in them. The presence of the code in such sentences might affect the research results and can be considered a limitation of the current data cleaning method.

Despite these limitations, the script achieved an average accuracy and F-score of over 90%, which we consider a good result for the purposes of this research.

3.1.2 Language detection

For language detection, which occurs after code lines were removed, we used two existing libraries: *langdetect*[2] and *pycld2 (Compact Langauge Detect 2)*[3]. Using both libraries improves detection accuracy — if one library fails to identify the language, the result from the other helps determine whether the prompt should be kept (if detected as English) or discarded.

The language identification process operates at the line level: each prompt is split into individual lines, and the language is identified for each line separately. If a line is detected as containing only programming code (as described in Section 3.1.1), it is marked as code and excluded from language detection. This approach helps avoid misclassifying non-English prompts as English due to the presence of English keywords commonly found in code.

The output of the language check is a tuple, where the first element represents the result from *langdetect*, and the second from *pycld2*. If the most common tuple across all lines in the prompt is either *("en", "en")*, *("en", "un")*, or *("un", "en")*, the prompt is considered to be in English. Any other combination results in the prompt being discarded.

3.2 Data exploration

As part of the data exploration process, we begin by extracting key information about the dataset. Before presenting the statistical overview, it is important to clarify the terminology used in this section. In interactions with ChatGPT, users submit inputs referred to as *prompts*, which the model uses to generate responses. A sequence of such prompt–response pairs within the same discussion thread creates a *conversation*. For the purposes of this

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

study, we focus exclusively on the user-generated prompts and disregard the responses produced by ChatGPT across all conversations.

3.2.1 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

To better understand the structure and variability of the dataset, we examine the distribution of conversation lengths across different data sources. Table 3.3 shows the basic statistics of conversation lengths across different data sources. Most datasets contain short conversations on average, with median lengths between 1 and 2 prompts, suggesting that interactions are typically brief. However, GitHub repositories stand out with a higher mean (21.38) and median (10.5), as well as the largest standard deviation (31.58), indicating a wider spread and the presence of longer discussions. This phenomenon can be explained by a low number of sharings (8 after non-English conversations removal), where each heavily influences the data.

Source	Mean	Median	Standard deviation
GH Commits	3.2	1	5.65
GH Issues	4.57	2	8.9
GH Discussions	4.66	2	9.52
GH Pull requests	3.36	2	7.94
GH code files	6.72	2	16.74
GH repositories	21.38	10.5	31.58
HackerNews	4	1	9.12

Table 3.3: Median, mean and standard deviation of the conversation lengths from different sources.

Figure 3.1 contains histograms that show how many conversations contain certain number of prompts, where the conversations with the maximum number of 30 prompts are shown. All the conversations that exceeded the number of 30 prompts were separated in the separate JSON files and their content was investigated and described below.

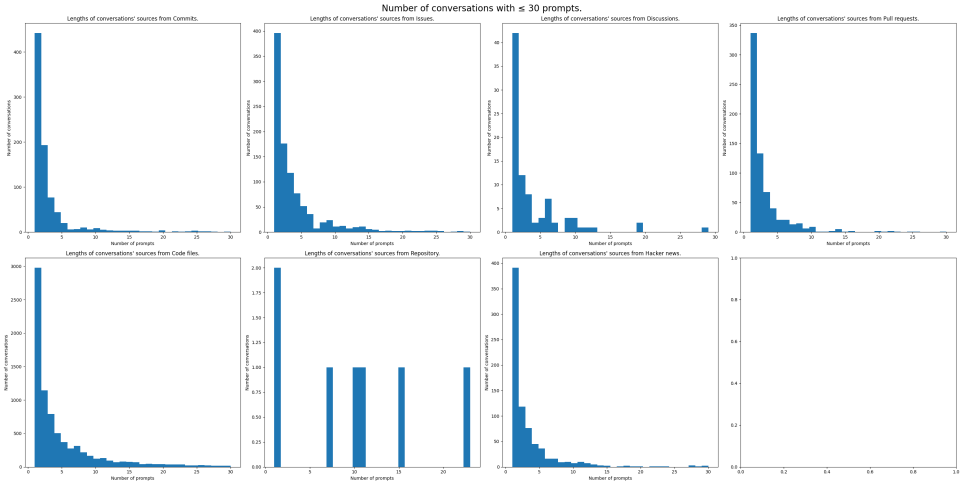

Figure 3.1: Lengths of conversations in histogram gathered from different sources.

Long conversations contents

To understand the reasons behind unusually long conversations, we manually reviewed conversations that exceeded a predefined length threshold. The most common patterns are summarised below:

- **Pasted content:** Users split large texts (e.g., code files, articles, books) into multiple prompts due to the prompt's character limits.
- **Iterative debugging:** Users refined requests or sought help with errors after initial code responses did not meet their expectations.
- **Programming assistance:** Users guided ChatGPT through multi-step coding tasks, testing and iterating on output.
- **Mentorship:** Users shared their own code for feedback, explanations, or improvement suggestions.
- **General discussion:** Some conversations focused on non-programming topics such as advice, opinions, or general discussions.

Average symbol/word count per prompt

The average number of symbols per prompt varied across sources, as shown in Figure 3.2. The x-axis represents the prompt position within a conversation, while the y-axis shows the average symbol count for that position. Since longer conversations are less common, later prompt positions often reflect fewer data points — leading to the prompt average pattern changes that can be visibly observed in the conversations sourced from GitHub code

files or Hacker News. Across majority of the sources, initial prompts typically have higher symbol counts, as they often introduce the problem in detail.

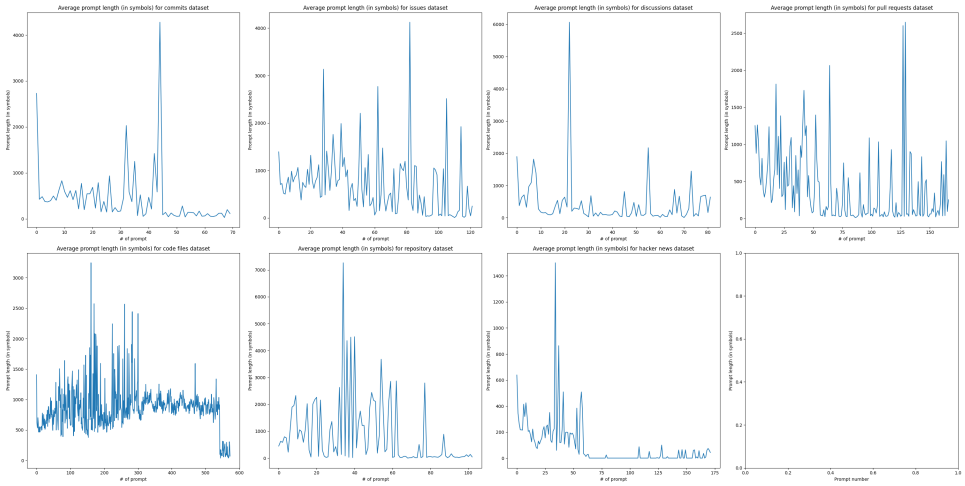

Figure 3.2: Average symbol count per prompt number for all conversation sources after code lines were removed.

To identify outliers, we defined dataset-specific cut-off thresholds set at two-thirds of the maximum observed average. Conversations containing prompts exceeding these thresholds were saved for further analysis.

Long prompt contents

Conversations containing unusually long prompts were saved and manually reviewed to understand their content. We identified several common reasons for the prompt length:

- **Pasted external content:** Users copied large texts from sources such as scientific articles, GitHub issues, or documentation, often asking for summaries, explanations, or audience-specific rewrites.
- **Code-related debugging:** Prompts included error messages, undocumented code, or comments not flagged as code, with users seeking help with debugging or improvement.
- **Detailed problem descriptions:** Some users provided extensive background to frame their query, including:
  - Highly specific requests or detailed issue descriptions;
  - Roleplay-style prompts (e.g., asking ChatGPT to act as a mentor or expert);

– Contextual or background information;
– Multiple follow-up questions or clarifications;
– Previously attempted (but unsuccessful) solutions;
– Long URLs or resource links;
– Lists of requirements or instructions;
– Desired output examples or target functionality.

Chapter 4

Methodology

The purpose of investigating how developers use ChatGPT in their projects is to gain a deeper understanding of the role the chatbot plays in developer-chatbot collaboration. This understanding can help improve the chatbot's ability to assist developers more effectively. To extract this knowledge, we applied n-grams, pattern mining, and topic modelling techniques, which are described in this chapter.

4.1 Data preparation

To enable further analysis, we pre-processed the cleaned data by lemmatizing and tokenizing each prompt.

The lemmatization process involves splitting each sentence into a list of elements such as words, punctuation marks, digits, or alphanumeric strings, and then converting each word to its lemma using the *lemmatize_sentence* function from the **pywsd** package [34]. The function attempts to convert each word into its base form. If the lemma is not found in WordNet, the function applies stemming as an alternative processing step.

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

- **Non-ASCII tokens**;
- **Hyperlinks** — token starts with *"http"*;
- **Stop words** — based on the `stop_words` list from the **nltk** package[1] [35];
- **Punctuation marks**;
- **Tokens containing digits**.

[1] Stop words list

The resulting lemmatized and cleaned lists of tokens are then used for the following tasks, including n-gram extraction, sequence mining, and topic modelling.

4.2 N-grams

N-grams is a sequence of n consecutive characters, words or other textual structures [36], where n refers to the number of elements in each group. These sequences are generated by sliding a window of size **n** over the text, preserving the original order of the elements. In this research, we focus on word-based n-grams. For example, the sentence *"That fox is chasing a rabbit"* can be tokenized into the following sequence of words: $\{that, fox, is, chasing, a, rabbit\}$. From this sequence, the following n-grams can be extracted:

- 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 n-gram extraction, the data is preprocessed as described in Section 4.1. This includes the removal of stop words — commonly used words that carry little semantic content *"that"*, *"is"* and *"a"*. In the example above, these words would be excluded, leaving only content-bearing words for n-gram construction.

N-grams are extracted using *ngrams()* function from **nltk** package, with the parameter n set to the desirable value. The extracted n-grams are then used to compute a frequency distribution, allowing us to sort them in descending order of occurrence. Such ranking helps highlight the most common lexical patterns found in the dataset.

For easier visualisation, we use the WordCloud package [37] to generate visual representations of the n-gram frequencies. In these visualisation, the font size of each m-gram corresponds to its relative frequency wjere more frequent patterns appear larger.

During the initial extraction process, we observed that some tri-grams and quadri-grams contained repeated instances of the same word (e.g., $(position, position, position)$). This effect is likely caused by the combination of lemmatization/stemming, stop word removal and imperfect code lines detection script, which can cause sequences of otherwise distinct words to collapse into repeated lemmas. To improve our results, we chose to exclude any n-grams that contain the same token repeated three or more times. This filtering step helps the quality of the analysis while preserving the patterns in the dataset.

4.3 Sequential pattern mining

Sequential pattern mining (SPM) is an essential part of knowledge discovery and data analysis often used in the analysis of medical treatment history [38], customer purchases [39] and digital clickstream [40]. SPM is used to find frequent patterns in large sequence databases.

An input for SPM is a set of sequences referred to as *sequence database*. Each sequence is an ordered set of *items*. A *pattern* is an ordered subsequence that occurs in at least one sequence in the database. *Frequency* defines the number of sequences that contain a pattern. Given an input sequence database, SPM involves finding frequent patterns that occus more than the specified threshold [41].

For example, if we have the sequences $\langle A, A, B, A, D\rangle$, $\langle C, B, A\rangle$ and $\langle C, A, C, D\rangle$, and the minimum frequency is set to two, the SPM extracts the following sequences: $\langle A, D\rangle$, $\langle B, A\rangle$ and $\langle C, A\rangle$.

In this research, we use **Seq2Pat** package [42] in order to extend sequential pattern analysis. N-grams, discussed in Section 4.2, extract sequential data of the set size. However, we expand this analysis using SPM in order to analyse frequent patterns seen in data that are longer than the longest n-grams we extract.

In order to extract frequent patterns, we initialise the *Seq2Pat* object with tokenized prompts as sequence database, set maximum span to 10, batch size to 10000 and discount factor to 0.2. We extract patterns using *get_patterns()* function, where minimum frequency is set to the length of the daatabase divided by 50. Once the sequence patterns are extracted, they are sorted in descending order and saved for the future analysis. The minimum frequencies for each data source were the following:

- GitHub commits: 54;
- GitHub issues: 90;
- GitHub discussions: 8;
- GitHub pull requests: 45;
- GitHub code files: 1069;
- GitHub repositories: 3;
- Hacker News: 60.

4.4 Topic modelling

Topic models are generative models that provide a probabilistic framework [43]. 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 [44]. 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 4.1. 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 [45]. 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 [46], 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

4.5 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[2] 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.

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

Chapter 5

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 N-grams, topic modelling and sequence mining of the prompts.

5.1 N-grams

N-grams are extracted in order to provide an overview of the most common lexical patterns found within the data. Data gathered from each source was analysed separately and the most common 1- to 4-grams were extracted. Table 5.1 shows interesting tri- and quadri-grams extracted from various data sources.

Source	n	Found n-grams
GitHub commits	3	shell script create, encode enclose result, avoid use sed, implement following feature
	4	script create change file, avoid use sed favor, task implement following feature
GitHub issues	3	th block th, valid string generation, state lookahead token, kernel patch instal
	4	td th block th, string generation metadata information, metadata information visit input, community edition line module
GitHub discussions	3	technology feasibility stable, operational feasibility challenge, clause potentially unfair, employee desirability mixed/high, open/close code environment

	4	technology feasibility stable employee, employee desirability high team
GitHub pull requests	3	gazebo sensor imu, plugin sensor accelerometer, deprecate pip enforce, possible replacement use
	4	coelemic cavity lumen subclassof, gazebo sensor imu plugin, sensor accelerometer parent origin, replacement use pip package, string sure want delete
GitHub code files	3	rdf type owl, restriction owl minqualifiedcardinality, passage book rewrite, break fourth wall
	4	rdf type owl objectproperty, rdfs subpropertyof owl topobjectproperty, george r r martin, book rewrite passage remove, break fourth wall include, wall include reference reader
GitHub repositories	3	fill height viewport, increase understanding universe, help reduce suffering, research educational institution
	4	height fill height viewport, leave position foreground scyscraper, reduce suffering increase prosperity, prosperity increase understanding universe, planet intellectual community train
Hacker News	3	capture position position, form line capture, tic tac toe, session user host, level level headline
	4	form line capture position, x currently form line, headline insert concat star, concat star schedule progn

Table 5.1: Interesting 3- and 4-grams extracted from the data.

The extracted n-grams reveal distinct linguistic and thematic patterns across different data sources. Prompts from GitHub commits and pull requests use task-oriented, direct and action-driven communication typical of development workflows. The prompts often focus on modifying, automating or improving code artefacts. In contrast, GitHub discussions contain planning-related n-grams like “clause potentially unfair” and “technology feasibility stable employee”, suggesting that contributors debate feasibility and desirability of the solution, not just technical correctness.

GitHub repositories show recurring themes in UI rendering (e.g.m CSS and Viewport references) as well as humanitarian, educational, and scientific discussions. However, prompts from GitHub issues and code files is influenced by embedded code or markup, such as HTML and RDF/OWL artefacts — indicating developers often seek help with specific technical

problems.

Hacker News conversations differ by a less technical, more playful approach. N-grams like “tic tac toe”, “form line capture position”, and “headline insert concat star” combine elements of game logic, syntax play, and UI behavior, typical of exploratory or playful community posts.

In conclusion, each platform reflects its users’ intent — ranging from practical coding to ethical reflection to creative exploration.

5.2 Topic modelling

5.3 Sequence mining

Chapter 6

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] C. Bou Rjeily, G. Badr, A. Hajjarm El Hassani, and E. Andres, "Medical data mining for heart diseases and the future of sequential mining in medical field," in *Machine learning paradigms: Advances in data analytics*, pp. 71–99, Springer, 2018.

[39] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proceedings of the eleventh international conference on data engineering*, pp. 3–14, IEEE, 1995.

[40] B. Requena, G. Cassani, J. Tagliabue, C. Greco, and L. Lacasa, "Shopper intent prediction from clickstream e-commerce data with minimal browsing information," *Scientific reports*, vol. 10, no. 1, p. 16983, 2020.

[41] S. Kadıoğlu, X. Wang, A. Hosseininasab, and W.-J. van Hoeve, "Seq2pat: Sequence-to-pattern generation to bridge pattern mining with machine learning," *AI Magazine*, vol. 44, no. 1, pp. 54–66, 2023.

[42] X. Wang, A. Hosseininasab, P. Colunga, S. Kadıoğlu, and W.-J. van Hoeve, "Seq2pat: Sequence-to-pattern generation for constraint-based sequential pattern mining," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 12665–12671, Jun. 2022.

[43] 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.

[44] 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.

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

[46] "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, 11–14, 16, 19, 21

LDA Latent Dirichlet Allocation. 19

LLM Large Language Model. 2

NLP Natural Language Processing. 4, 5

SPM Sequential pattern mining. 18

Reflection on research process

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

Appendix A

Appendix

A.1 N-grams WordClouds

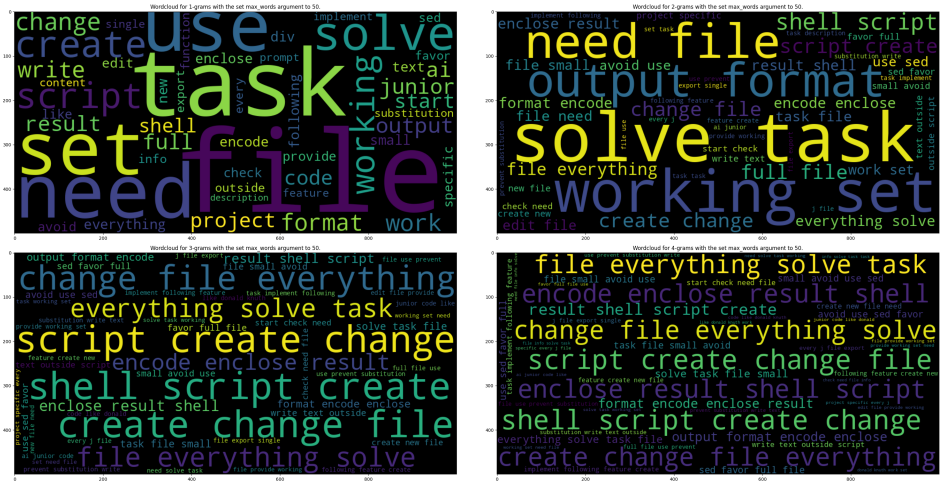

Figure A.1: N-grams from conversations sourced from GitHub commits.

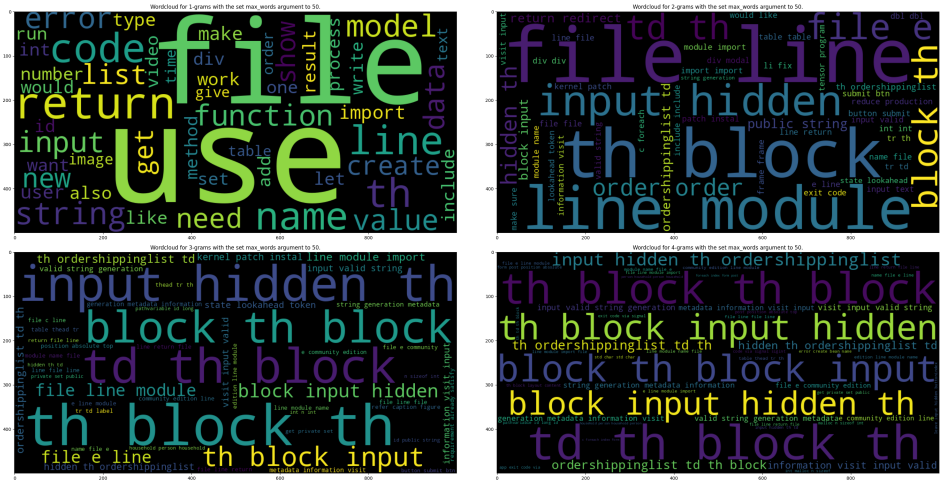

Figure A.2: N-grams from conversations sourced from GitHub issues.

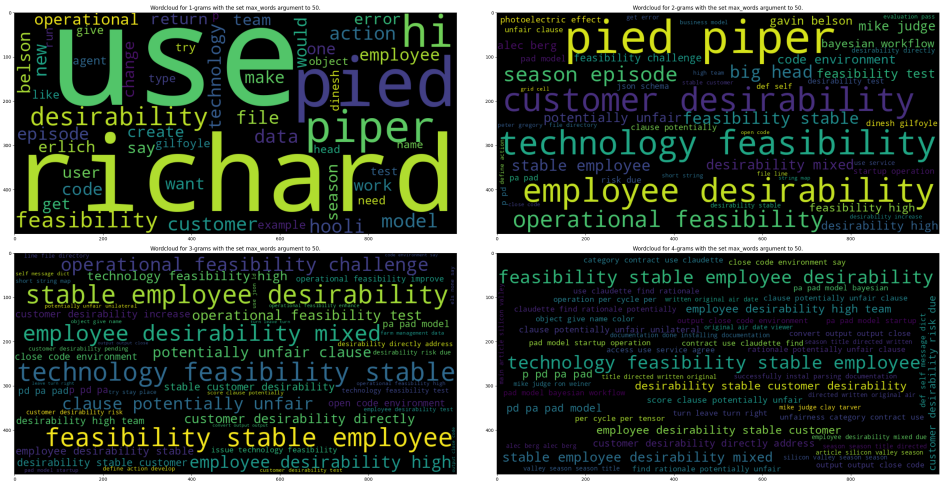

Figure A.3: N-grams from conversations sourced from GitHub discussions.

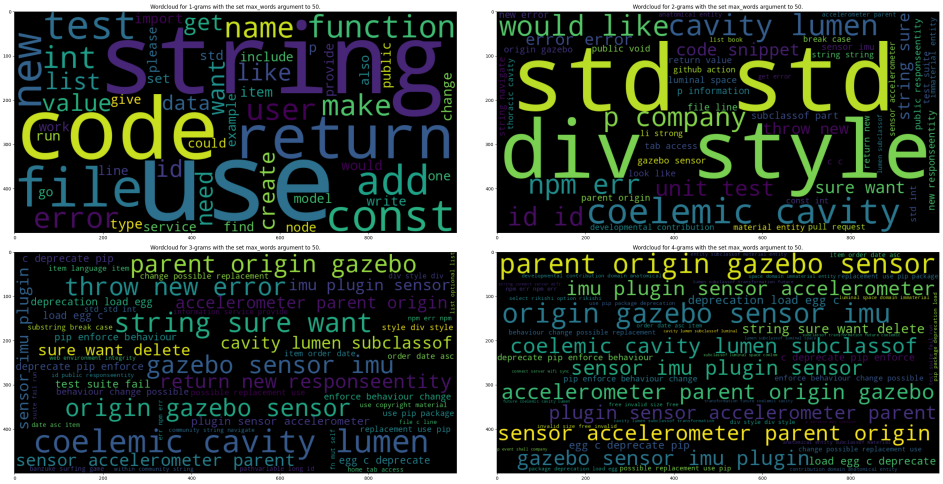

Figure A.4: N-grams from conversations sourced from GitHub pull requests.

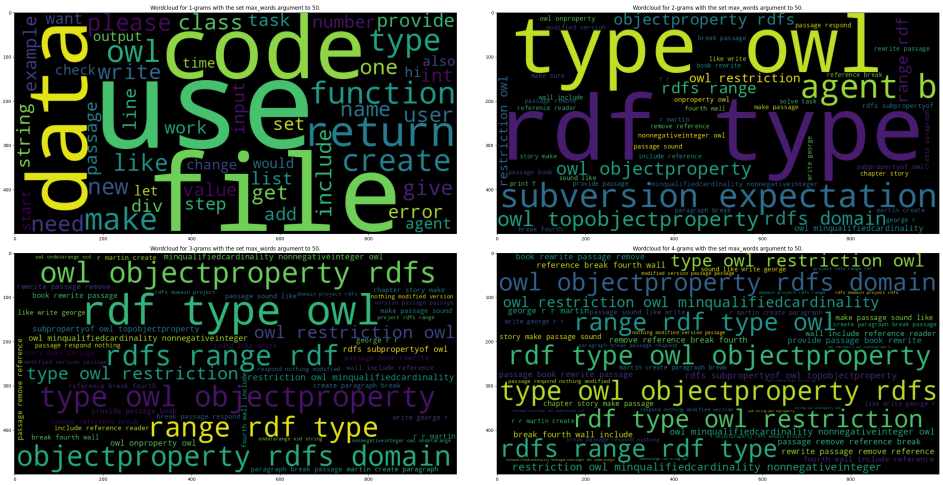

Figure A.5: N-grams from conversations sourced from GitHub code files.

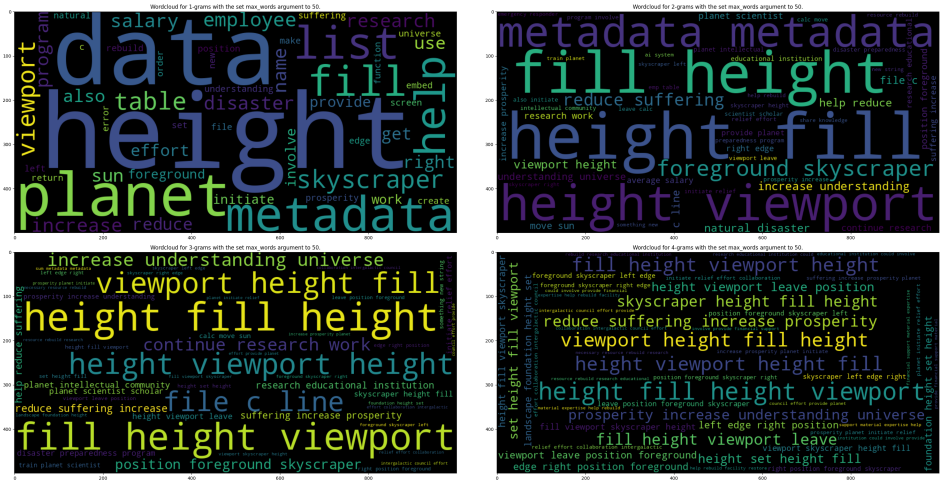

Figure A.6: N-grams from conversations sourced from GitHub repositories.

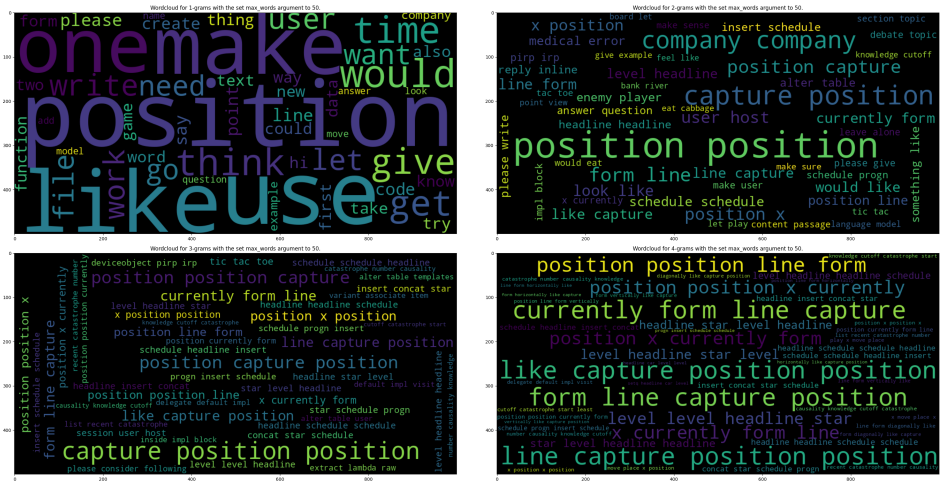

Figure A.7: N-grams from conversations sourced from Hacker News.