DevGPT: An analysis on conversations between developers and ChatGPT

Kiran John

Seamus Riordan

Skylar Shao

BACKGROUND

The DevGPT dataset is a curated dataset containing 17,913 ChatGPT prompts related to various software development artifacts – which is grouped by Hacker News threads, pull requests, issues, discussions, commits, and source code. The data is further subdivided into six different snapshots from July to August 2023. In this study, we aim to explore and answer three key research questions:

RQ1. What types of issues (bugs, feature requests, theoretical questions, etc.) do developers most commonly present to ChatGPT?

RQ2. How do the answers provided by ChatGPT vary between models (length, content, tone, etc.)?

RQ3. Can we reliably predict the author of a given piece of text from the content of the message?

Through this paper, we demonstrate our methodology, present our results, and discuss our findings.

METHODOLOGY AND RESULTS

Data Pre-Processing

For each research question, we engage in similar steps to set up the data for analysis. Since the DevGPT files are in a json format and separated by their respective artifact, our first task is to join them together to be able to conduct a comprehensive analysis. Using the os and json libraries, we can parse out embedded columns within each file into a larger master file. For RQ1 and RQ2, we use the data from 'snapshot_20231012', while we use data from all the snapshots for RQ3. The main purpose of using the entire dataset for RQ3 is to enable our prediction model to have increased performance. Note that we also do some extra exploration of the dataset, but we chose to omit this to focus on the questions we found most interesting.

RQ1. What types of issues (bugs, feature requests, theoretical questions, etc.) do developers most commonly present to ChatGPT?

The objective of our first question is to categorize and quantify the most common types of issues developers present to ChatGPT. To do so, we follow a clear and structured approach. We adopt the pre-processing mentioned in the previous section, combining the datasets from 'snapshot_20231012'. We then remove duplicate prompt and answer pairs to mitigate redundancy within our results. Additional data cleaning includes removing punctuation, extra spaces, and symbols within both the prompts and answers. To simplify text analysis, we defined five issue categories based on keyword occurrences in developer prompts and responses:

- *Bug Reports:* Identified using the keywords "error", "bug", "crash", "exception", "fail", "issue", "fix"
- *Feature Requests:* With the keyword's "feature", "enhance", "add support", "improve", "implement"
- *Theoretical Questions:* Includes "explain", "theory", "why", "difference", "how does"
- *Performance Issues:* Keywords like "slow", "optimize", "bottleneck", "latency", "performance"
- *Code Help:* Having the keywords "how to", "example", "sample", "syntax", "help", "debug"

To find the number of occurrences for each keyword, we simply count the appearances of a given phrase within either the 'Prompt' or 'Answer' columns. After gathering the counts, we aggregate by a defined 'Issue Type' category (Figure 1).

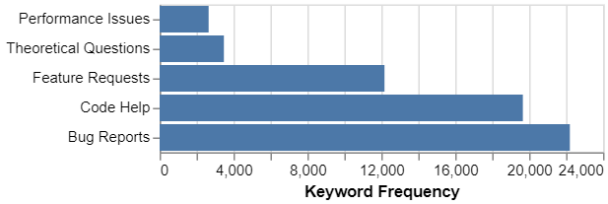

Figure 1. Types of issues presented to ChatGPT, ordered by ascending frequency.

As we see from the chart, bug reports are the most common issue type, which indicates that developers are frequently seeking assistance in debugging and resolving errors within their code. Code help is also a prevalent issue, which may also be related to bug reports, but also extend to general syntax. Though not as present as the other two issue types, it

appears that developers are also interested in adding new functionalities or improving existing implementations with feature requests. Theoretical questions are relatively low, which suggests that developers are primarily using ChatGPT for practical, problem-solving tasks rather than extensive discussions. Performance issues are the least found issues, which means most users are not concerned with improving code efficiency (and instead prefer functional code).

RQ2. How do the answers provided by ChatGPT vary between models (length, content, tone, etc.)?

In the second research question, we extend our initial analysis into comparing ChatGPT models and the text within each conversation. We apply the same methods used in RQ1: combine datasets from 'snapshot_20231012', remove duplicate prompt/answer pairs, and clean the text in both columns to enable robust text analysis. Additionally, we remove all entries containing non-English text to strictly focus on English conversations. We can then plot the number of conversations by a given model (Figure 2).

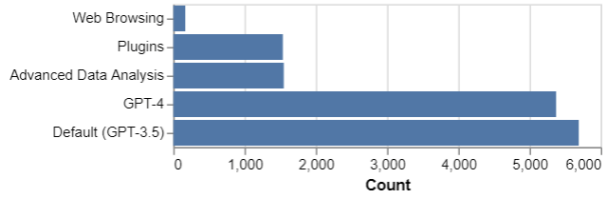

Figure 2. Number of conversations with ChatGPT, sorted in ascending order by model

As we observe in the figure above, the default model (GPT-3.5) and GPT-4 are the most used models by developers. Developers rarely use ChatGPT for its web browsing model but occasionally use it for plugins and advanced data analysis. We can also break down further variations between models by the word count of each answer as seen in Table 1.

Model Used	Mean	Median	Std. Deviation
Advanced Data Analysis	200.347966	164.0	163.157819
Default (GPT-3.5)	166.418981	133.0	136.595871
GPT-4	161.968727	140.0	121.575601
Plugins	176.986979	154.0	122.273996
Web Browsing	224.562874	182.0	169.818704

Table 1. Summary statistics on word counts of a ChatGPT model

Web browsing and Advanced Data Analysis answers are much longer than other models, which can be attributed to their comprehensive nature. The more common models, GPT-3.5 and GPT-4 have similar average word counts in their responses, but the response length for GPT-4 has the least variance between answers. This would imply that GPT-4 is a much more consistent model compared to the others. We can also perform a simple text analysis to examine some more differences between each model. To do so, we first need to remove prepositions and other frequently appearing words, as they will be the most common words if not removed. Afterwards, we can apply vectorization via TF-IDF and find the most frequent words by model. In brief, the top five words per model (that are in the English dictionary) are as follows:

- *Advanced Data Analysis:* "data", "like", "file", "finished", "function"
- *Default (GPT-3.5):* "code", "coding", "use", "file", "data"
- *GPT-4:* "code", "function", "use", "data", "using"
- *Plugins:* "like", "function", "code", "students", "use"
- *Web Browsing:* "used", "slack", "need", "use", "code"

Quite clearly, "code" is a very common word within all answers, which would stem from ChatGPT providing code back to the user. GPT-4 and Plugins seem to focus more heavily on specific functions while Advanced Data Analysis and the Default model are related more to files and data. Web browsing also uses "slack" quite commonly, which we do not know the exact context for. Since there are similar words between some models, our last exploration in this research question employs the use of clustering to determine general similarities between models. Specifically, we use KMeans clustering with the from scikit learn. The results can be seen in Table 2 below. As we can observe, GPT-3.5, GPT-4, and Plugins seem to have similar content in their answers, while Advanced Data Analysis and Web Browsing appear to have unique words.

Model Used	Cluster
Advanced Data Analysis	0
Default (GPT-3.5)	2
GPT-4	2
Plugins	2
Web Browsing	1

Table 2. Results of clustering on ChatGPT models

RQ3. Can we reliably predict the author of a given piece of text from the content of the message?

Our final research question employs natural language processing (NLP) in combination with random forest (RF) classification to predict a developer's identity from the content of their writing. Like in the previous RQs, data across all the file categories are merged into a single working data frame. Data across all the snapshot dates are also combined – given the expected difficulty of performing this task accurately, we want to provide the model with as much information as possible. Furthermore, the various columns that user text might be found in (e.g. comments, commit messages, discussion threads) are combined into a single text column. This column was cleaned and tokenized using a preprocessing function to give a data frame containing an "author" column, a "type" (or category as described above) column, and a "text" column containing tokens. There are

1883 unique usernames in the "author" column, with 18714 total posts.

Training and test sets are created from this dataframe and a vector model (Word2Vec) is applied to the training data. Using this model, arrays of training and test vectors are created by first creating word vectors for each piece of text and then averaging these to create a vector that represents the whole message. This is done to create vectors of equal length for modelling, and because the individual word vectors are not relevant to this task. This specific approach is adapted from a blog posted on Medium (Valeti). A random forest model (RandomForestClassifier) is then trained on these vectors with "Author" as the predicted label, and model accuracy is calculated on the hidden test data

Initially, this procedure is conducted using default values for the Word2Vec and RandomForestClassifier classes. The model performs surprisingly well, correctly identifying the author in 86.91% of the withheld rows. However, we know modifications can be made and several steps are being taken to improve performance. Rows identified by authors with less than 5 total posts are removed (which eliminates 8% of the original dataset) as we suspect that the identities of authors with too few posts would be challenging to predict. This leaves 17217 posts from 1105 unique authors. The Word2Vec model is adjusted to use a 'min_count' of 1 instead of the default 2 (as unique words are expected to be valuable for identifying authors) and two other parameters, vector_size' and 'window', are tuned using 5-fold cross validation. A randomized search cross validation (RandomizedSearchCV) function is used to tune the random forest classifier as well, providing the following optimal parameters: 'n_estimators': 20, 'max_features': 'sqrt', 'max_depth': 15, 'criterion': 'log_loss'. This optimization only provides a modest improvement to the original model, achieving 92.33 % accuracy on the test set.

The effect of filtering the dataset by some minimum number of posts per user is further explored to answer the secondary question: how many posts per user do we need to predict to a very high level of accuracy? Iteratively refitting the model to these filtered datasets shows a dramatic increase in accuracy when limiting to higher-volume authors (Figure 3).

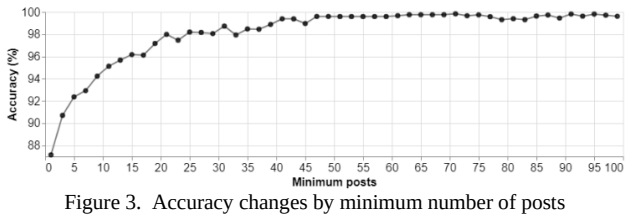

Figure 3. Accuracy changes by minimum number of posts

Finally, for comparison, "Type" is predicted instead of "Author" using the text tokens. The dataset used for this does not include the "Type" tokens in the training data, nor is it limited to authors with a minimum number of posts. The previous optimization procedure is used, and the model correctly predicts the source category of text in the test set with 97.60 % accuracy.

DISCUSSION

With the methods we conduct during our analysis, we can provide concrete conclusions for each of our research questions.

In RQ1, we explore the types of issues which developers present to ChatGPT. After setting categories to classify various keywords in, we find that bug reports and code help consist of the bulk of user issues. As mentioned, this suggests that many developers are currently utilizing ChatGPT for fixing their code if they run into errors or bugs during implementation. It is much less common for developers to present questions that are more theoretical or performance-related in nature.

When we look at RQ2, our goal is to draw comparisons between different ChatGPT models and offer insights in their answer content. What we find is that GPT-3.5 and GPT-4 are quite similar in answer length and are also grouped together when applying clustering. Other models exhibit more stark differences, with higher word counts and dissimilar content. What we surmise from this is that GPT-3.5 and GPT 4.0 are both reliable choices for assisting developers if they are concerned with code or data.

RQ3 appears to have been affirmatively answered for this data set. Without much treatment beyond tokenizing text, ~87% of authors can be correctly identified from the content of their posts. With some optimization this accuracy rises slightly, but most notably, if the dataset is filtered to users with significant numbers of posts, we can achieve an extremely high degree of accuracy. While this requires some "foreknowledge" in the sense that we must assume all input content was posted by one of these high-posting users, this assumption is reasonable if we are looking at communities where users are very active. The high predictive accuracy can be explained by the nature of the data set – all the text is within the context of developers using ChatGPT to help with specialized topics. The technical language expected in this content is likely well suited for identifying specific users.

In general, we answer all three of our proposed research questions with substantiated results. However, we recognize that there are many areas that are still worth examining. To extend this analysis, we want to conduct more extensive text analysis and consider other factors, such as time, when evaluating the prompts and answers provided by developers. With the comprehensive nature of the DevGPT dataset, these steps are feasible, and we hope to explore more facets of the dataset in the near future.

REFERENCES

[1] Valeti, Dilip. "Classification using Word2vec". Medium, 2021. (https://medium.com/@dilip.voleti/classification-using-word2vec-b1d79d375381)