# Nature of Developer-ChatGPT Interactions

Ming Xiao Yuan
*Department of Computer and Software Engineering*
*Polytechnique Montreal*
Montreal, QC, Canada
ming-xiao.yuan@polymtl.ca

Reetesh Dooleea
*Department of Computer and Software Engineering*
*Polytechnique Montreal*
Montreal, QC, Canada
reetesh-chandranath.dooleea@polymtl.ca

***Abstract*—This study aims to elucidate the dynamics of Developer-ChatGPT interactions by analyzing a substantial dataset from the DevGPT public GitHub repository. This repository is a rich compilation of snapshots capturing numerous exchanges between developers and ChatGPT throughout the second part of the year 2023. It encompasses a wide array of queries posed by developers and the corresponding responses provided by ChatGPT. The primary objective of this research is to gain deeper insights into the nature of these interactions. By doing so, it seeks to offer a nuanced understanding of how developers typically engage with ChatGPT. This exploration is not only pivotal in shedding light on the current state of developer-AI interactions but also crucial in informing future developments and enhancements in the realm of AI-assisted programming and problem-solving.**

***Index Terms*—Natural Language Processing (NLP), Machine Learning (ML), Text Classification, Data Mining, Data Analysis.**

## I. Introduction

The MSR 2024 Mining Challenge, an annual data mining competition, presents a unique dataset each year centered around a topic of significant interest, inviting participants to explore and analyze it through their chosen research questions. This year, the spotlight is on the burgeoning field of large language models (LLMs) like ChatGPT. The challenge revolves around the DevGPT dataset, a compiled collection of developer conversations with ChatGPT. This dataset is rich with various prompts from developers and ChatGPT's responses, including code snippets. The aim of DevGPT is to facilitate an in-depth exploration of how developers interact with ChatGPT, providing insights into the nuances and broader implications of these AI-mediated exchanges in the realm of software development and beyond.

In this paper, we are driven by the motivation and objective to analyze and characterize the nature of developer interactions with ChatGPT. Thus, we aim to do perform an empirical study to answer the two following research questions:

- **RQ-1: What types of issues (bugs, feature requests, theoretical questions) do developers most commonly present to ChatGPT?**
- **RQ-2: What is the typical structure of conversations between developers and ChatGPT? How many turns does it take on average to reach a conclusion?**

In addressing our first research question, we employed two distinct methods. The initial approach, termed *Simple Issue Categorization*, is a straightforward technique for classifying issues (encompassing both questions and answers) as *Bug*, *Feature Request*, *Theoretical Question* or *Other*. This method primarily relies on identifying specific keywords within the questions and answers, guiding the classification process. However, our findings revealed a significant limitation of this approach, with a majority (77.6%) of the issues being categorized as *Other*, indicating a lack of precision in this method.

Recognizing the inadequacies of the *Simple Issue Categorization*, we progressed to a more sophisticated approach which we name as *Issue Categorization via Machine Learning Model*. This method involved training a machine learning (ML) model on a dataset of labeled issues spanning all categories. Once trained, this model was then utilized to predict the classifications of issues within the DevGPT snapshots. The results from this ML-based approach marked a substantial improvement in accuracy compared to the initial method, with only 20.2% of issues being classified as *Other*. This stark contrast underscores the enhanced precision and reliability of using a machine learning model for such categorization tasks in our research.

Our research question 2 is linked to our findings from research question 1. In other words, after classifying conversations into the four defined issue categories, we store the results in a dictionary. This data structure is served as input to our research question 2, which once again, aims to understand the average structure of conversations between developers and ChatGPT. To achieve our results, our approaches primarily relied on intensive data mining and processing techniques. For instance, we iterated through our organized dataset, employing various statistical operations to extract meaningful insights and derive our conclusions to this research question.

As for the coding resources, all of our code created and

established for mining the DevGPT dataset to answer our research questions can be found on our GitHub repository. At the same time, it is essential to highlight that we opted for Jupyter Notebook as our primary coding environment, finding it user-friendly as well as more authentic and readable for our research purpose. The programming language employed was Python 3. Furthermore, snapshots of the dataset used in this process are available on DevGPT repository.

## II. Background Concepts

To begin with, there are a few software background concepts that needs to be clarified in order to better understand this paper.

The first concept is Natural Language Processing (NLP). It's a branch of artificial intelligence (AI) that focuses on enabling computers to understand, interpret, and respond to human language in a way that is both meaningful and useful. NLP combines computational linguistics—rule-based modeling of human language—with statistical, machine learning, and deep learning models. These technologies enable computers to process human language in the form of text or voice data and to understand its full meaning, complete with the speaker or writer's intent and sentiment.

Another concept is machine learning (ML). It's a subset of artificial intelligence that involves the development of algorithms and statistical models that enable computers to perform tasks without explicit instructions. Instead, these systems learn and make decisions based on patterns and inferences from data.

Text classification is a process in machine learning and natural language processing where a computer system categorizes pieces of text into one or more predefined categories or classes. This technique involves analyzing and interpreting the content of text data and assigning it to appropriate categories based on its content.

Data mining is the process of extracting valuable insights, patterns, and knowledge from large sets of data, utilizing methods at the intersection of machine learning, statistics, and database systems. It involves the analysis of data from various perspectives and summarizing it into useful information, which can be used to increase revenue, cut costs, improve customer relationships, reduce risks, and more.

Finally, data analysis is the process of inspecting, cleansing, transforming, and modeling data with the goal of discovering useful information, informing conclusions, and supporting decision-making. It involves applying statistical, logical, and analytical techniques to data to identify trends, patterns, and relationships, which can then be used to solve problems or make informed decisions.

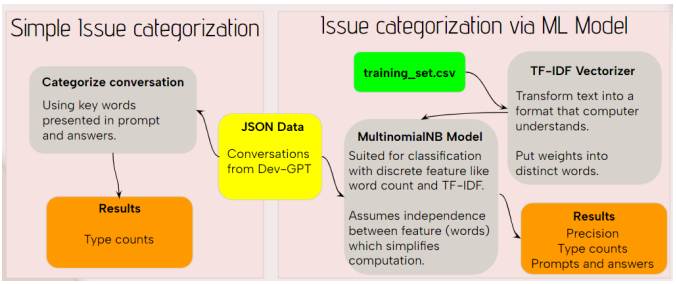


Fig. 1. Representation of both categorizations

```
In [3]:
def categorize_conversation(prompt, answer):
    prompt = prompt.lower()
    answer = answer.lower()

    if "bug" in prompt or "error" in answer:
        category = "Bug"
    elif "feature" in prompt or "implement" in answer:
        category = "Feature Request"
    elif "how to" in prompt or "why" in answer:
        category = "Theoretical Question"
    else:
        category = "Other"

    return category
```

Fig. 2. Underlying decision logic of Simple Categorization

## III. Methodology

### A. RQ-1: What types of issues (bugs, feature requests, theoretical questions) do developers most commonly present to ChatGPT?

The methodology for both *Simple Issue Categorization* and *Issue Categorization via ML Model* is depicted in Figure. 1, which presents a streamlined overview of each categorization approach. The process begins with data or conversations in JSON format from DevGPT, focusing on extracting specific values: prompts from developers and responses from ChatGPT. This data extraction is a crucial preliminary step before the information can be input into either categorization systems. To maintain simplicity, this paper will not further explore the specifics of mining data from JSON files.

#### 1) Simple Issue Categorization:

As mentioned beforehand, this type of categorization first involves in reading the different prompts and answers from ChatGPT. DevGPT uses a JSON format to store the different data. In order to fetch the questions from developers and answers from ChatGPT, we had to first mine the JSON files and then preprocess our data in order to fetch the desired data.

After completing the data extraction step, the information is then processed through the *Categorize Conversation* algorithm, as illustrated in Figure. 2 which includes sample

code for the ‘categorize conversation‘ function. This function operates by searching for specific keywords within the developers’ prompts and ChatGPT’s responses. To categorize a conversation as a *Bug* issue, the algorithm looks for the keyword ”bug” in the developer’s prompt and ”error” in ChatGPT’s response. Similarly, an issue is labeled as a *Feature Request* if the keyword ”feature” is present in the prompt and ”implement” appears in the response. If the prompt contains ”how to” and the response includes ”why,” the issue is categorized as a *Theoretical Question*. In cases where none of these specific keywords are detected, the issue is classified as *Other*.

After the categorization of all issues, the algorithm’s output provides a count for each category. By examining which category has the highest count, we can address our initial research question 1. This analysis allows us to understand the predominant type of issues encountered, offering insights into the areas that may require more attention or improvement.

*2) Issue categorization via Machine Learning Model:*

The *Simple Issue Categorization* method served as an initial test to determine the distribution of issues across various categories using a straightforward approach. However, it became apparent that this method did not ensure the accuracy and quality of the categorization predictions. To address this limitation, a decision was made to employ a machine learning (ML) model. This model is designed to more reliably predict whether an issue falls into specific categories: a *Bug*, a *Theoretical Question*, a *Feature Request*, or as *Other*. The use of a ML model aims to enhance the precision and reliability of the issue categorization process.

TABLE I
EXAMPLE OF CONTENT OF TRAINING SET OBTAINED FROM CHATGPT

| **Text** | **Label** |
|---|---|
| Write a function to calculate Fibonacci numbers. I’m getting an error in my recursion logic. Can you help me fix it? Error: StackOverflow at line 10. | Bug |
| How can I implement a real-time chat feature in my web application using WebSockets? I need a basic implementation guide. | Feature Request |
| What’s the difference between a LinkedList and an ArrayList in Java? I’m trying to understand which one to use for my project. | Theoretical Question |
| How important is it to learn about algorithms and data structures for a software engineer? | Other |

The process involves several interconnected components, with the training set being a primary element. As depicted in Table. I, which shows a segment of the training set, there are two main columns: ’Text’ and ’Label’. The ’Text’ column contains a variety of questions posed to ChatGPT, representing the kind of inquiries it typically receives. These questions are then categorized in the ’Label’ column into one of four distinct categories. For the creation of this training set, ChatGPT itself was utilized to generate a range of questions that fit into each of these categories. Subsequently, these questions were labeled into the four designated categories. This process echoes the manual approach that would have been used prior to the development of ChatGPT, where each question had to be individually classified into a specific category by a person. This approach lays the groundwork for training the machine learning model, providing it with the necessary data to learn how to accurately categorize questions.

The TF-IDF (Term Frequency-Inverse Document Frequency) Vectorizer is a key feature extraction tool in text analysis and Natural Language Processing (NLP). It transforms text into a numerical format understandable by machine learning algorithms. The process involves two components: Term Frequency (TF), which measures how often a term appears in a document, and Inverse Document Frequency (IDF), which gauges the term’s importance across a set of documents. TF is the ratio of the number of times a term appears in a document to the total number of terms in the document, while IDF is the logarithm of the ratio of the total number of documents to the number of documents containing the term. This dual approach helps balance the relevance of terms within individual documents and their significance across a larger corpus, ensuring that the text data is effectively processed for machine learning applications.

The Multinomial Naive Bayes (MultinomialNB) model is a probabilistic learning method commonly used in Natural Language Processing (NLP) for text classification. It operates under the principle of the Bayes theorem and assumes independence between the predictors. The multinomial aspect of the model refers to its ability to handle data that can be represented as frequency counts or occurrence counts, a feature particularly useful in text classification where data are often word counts or TF-IDF representations. We chose the MultinomialNB model in conjunction with the TF-IDF Vectorizer for our analysis due to its effectiveness in handling high-dimensional sparse data, which is typical of text. The TF-IDF Vectorizer transforms raw text into a numerical format that encapsulates the importance of words within documents and across a corpus, creating an ideal input for the MultinomialNB model. This combination is particularly powerful as it leverages the strengths of both the vectorizer and the classifier: the vectorizer’s ability to highlight significant words and reduce dimensionality, and the classifier’s proficiency in handling frequency-based data, making it a robust choice for categorizing the nature of developer interactions with ChatGPT in our study.

In the final stage of our machine learning (ML) model’s operation, it outputs three distinct elements. First, it provides a count for each category, offering a quantitative measure of how many issues fall into each of the predefined categories. Second, the model generates a dictionary that contains the

prompts and answers associated with each category. This dictionary serves as a detailed reference, mapping specific types of interactions to their respective categories and will be used later in answering research question 2. Lastly, the model calculates a precision count, which is a metric that evaluates the accuracy of the model in categorizing the issues correctly. The detailed results and analysis of these outputs, including the effectiveness and precision of the ML model, will be discussed in Section 4 of the paper.

### *B. RQ-2: What is the typical structure of conversations between developers and ChatGPT? How many turns does it take on average to reach a conclusion?*

In order to make our challenge even more interesting and intriguing, we decided to connect our research question 2 with our research question 1. In other words, the input or dataset used for this research question is based on the dictionary produced from our model from research question 1 that maps every prompt and answer to a category. In that way, we have four keys in our dictionary, i.e., *Bug*, *Feature Request*, *Theoretical Question* and *Other*, where the value of each key refers to a list of prompts and answers. From this classification of conversations into different categories, we are then able to perform several statistical operations on this data in order to answer our second research question which is to understand the typical structure of conversations between developers and ChatGPT.

The methodology for this part of our study is straightforward. For each analysis, we aim to create visualization graphs to illustrate our results with more specificity. Hence, to achieve this requirement of ours, we mainly employ extensive data processing and manipulation techniques from our new dataset, including the use of deeply nested *for loops*, *dictionaries* and other methods, to create diverse data structures. These structures are then utilized and fed as inputs to generate visualization graphs such as pie charts and bar charts.

## IV. Results

### *A. RQ-1: What types of issues (bugs, feature requests, theoretical questions) do developers most commonly present to ChatGPT?*

#### *1) Simple Issue categorization:*

The outcomes from this section are visually depicted in a pie chart in Figure. 3. This chart represents the analysis of a total of 153,556 conversations extracted from JSON files. According to the chart, *Theoretical Question* make up 2.9% of all conversations, *Feature Request* account for 9.6%, and *Bug* represent 9.9%. Notably, the category labeled *Other* encompasses the majority, with 77.6% of all

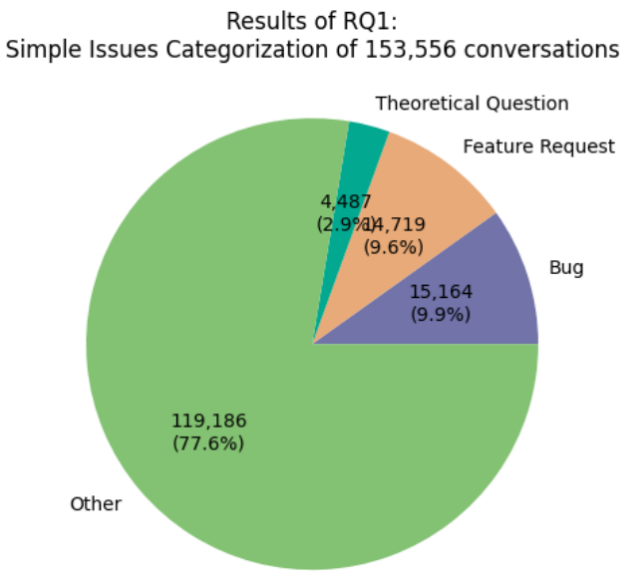


Fig. 3. Results of Simple issue categorization

```
Validating model.
        Accuracy: 0.73
        Precision: 0.73
        Recall: 0.73
        F1 Score: 0.73
Model validation complete.
```

Fig. 4. Machine Learning model precision

conversations falling under this classification. As previously mentioned, this method primarily offers a preliminary insight into the distribution of issues across different categories. It serves as a foundational guideline for the more advanced *Issue Categorization via ML Model*, providing an initial understanding of the issue landscape that the ML model will further investigate and categorize with greater precision.

#### *2) Issue categorization via Machine Learning model:*

Figure. 4 above showcases the four metrics used to evaluate the precision of the machine learning model. Accuracy, the first metric, indicates the overall rate at which the model correctly predicts categories, with our model achieving 73% accuracy. This suggests a strong general reliability in its predictions. Precision measures the ratio of correctly predicted positive observations to total predicted positives, highlighting the model's ability to minimize false positives and ensure relevance in its categorization. Recall, or sensitivity, assesses the model's capability to identify all relevant instances within the dataset, ensuring comprehensive coverage. Finally, the F1 Score, which is the harmonic mean of precision and recall, provides a balanced view of the model's performance, considering both its precision and thoroughness. The combination of these metrics, particularly

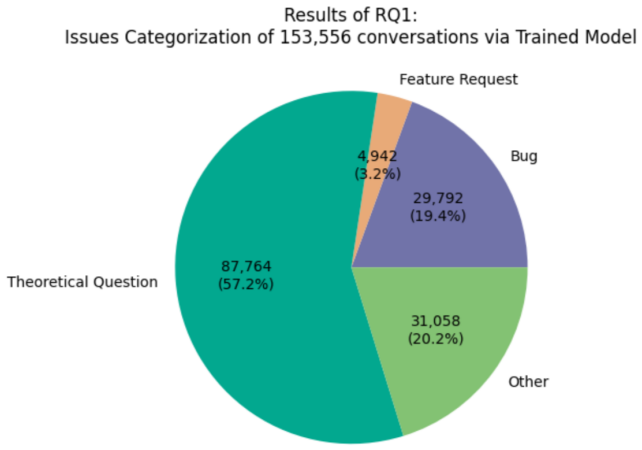


Fig. 5. Results of Issue categorization via Machine Learning model

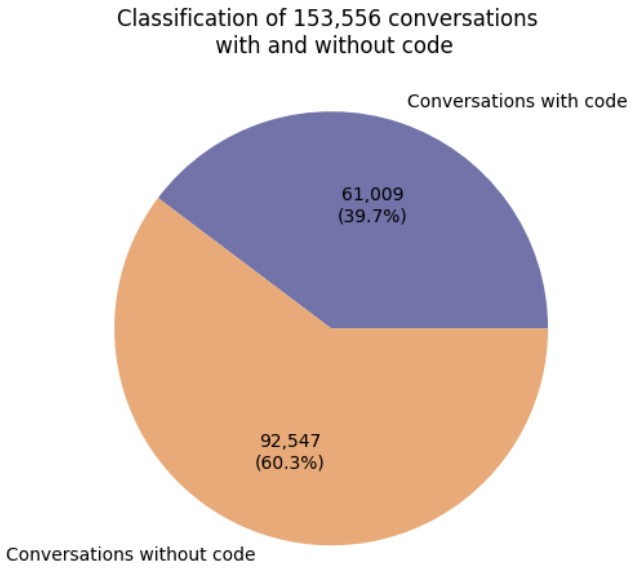


Fig. 6. Results of conversations with and without code

with a 73% accuracy rate, supports the conclusion that our model predicts categories fairly well, balancing accuracy with precision and recall, making it a reliable tool for categorizing issues in the DevGPT JSON files. For the purpose of this study, the training set was divided into two parts: 80% for training the model and 20% for testing it. The results of the four metrics are based on the model's ability to accurately predict the categorization of this 20% test data segment. According to the figure, the model successfully predicts the correct categories for 73% of the test set. This level of accuracy is considered fairly good, especially considering the limited amount of data in the training set. The 73% accuracy rate provides sufficient confidence in the model's capability to categorize issues in the JSON files from DevGPT. Thus, the model is deemed adequate for predicting the nature of issues based on the available data.

Figure. 5 presents a pie chart illustrating the results from the *Issue Categorization via Machine Learning Model* approach, demonstrating a notable improvement in accuracy compared to the *Simple Issue Categorization*. The ML model's results show a different distribution: *Feature Request* constitute 3.2% of the issues, *Bug* account for 19.4%, *Theoretical Question* make up a significant 57.2%, and *Other* are reduced to 20.2%. This contrasts with the *Simple Issue Categorization*, where *Bug* and *Feature Request* were almost equal (9.6% and 9.9%, respectively), whereas the ML model reveals a higher incidence of *Bug* compared to *Feature Request* (19.4% vs. 3.2%). Additionally, the substantial decrease in issues categorized as *Other* (from 77.6% in the simple approach to 20.2% in the ML model) indicates a more precise categorization. These results suggest that the ML model is significantly more effective in accurately identifying and categorizing issues, reducing the ambiguity and generalization seen in the simpler approach.

### *B. RQ-2: What is the typical structure of conversations between developers and ChatGPT? How many turns does it take on average to reach a conclusion?*

#### *1) Proportion of conversations including code snippets:*

Our initial emphasis was on examining the proportion of conversations that included fragments of code snippets within the discourse. More precisely, we wanted to quantify the percentage of conversations with and without code. As a result, a preliminary observation as illustrated in Figure. 6 reveals that approximately 39.7% of conversations, which is 61,009 out of 153,556 conversations, usually include code references. This analysis highlights the presence of technical aspects and elements within conversations between developers and ChatGPT which involves lots of code exchanges. As for the remaining 92,547 conversations not involving code snippets, this can be comprehended by the fact that *Theoretical Question* is the most common kind of issue developers present to ChatGPT, as concluded in research question 1. Since *Theoretical Question* is usually conceptual or abstract in nature, developers tend to explore ideas to learn and discover new subjects, rather than solving practical programming problems or issues. Consequently, it is reasonable to assert that *Theoretical Question* usually does not include code snippets as references which explains the reason as to why 60.3% of conversations commonly lack blocks of code.

#### *2) Average number of exchanges in a conversation per issue category:*

Next, our attention turned to comprehending the average number of turns or exchanges occurring in a typical

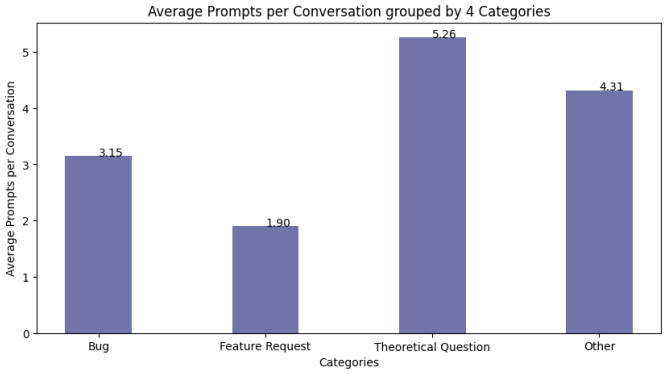


Fig. 7. Results of average exchanges per conversation for each issue category

conversation between developers and ChatGPT. An exchange refers to one question raised from a developer and an answer provided by ChatGPT in return. Hence, to achieve this, data analysis techniques were employed to categorize the average number of exchanges within our four defined categories. In simpler terms, this analysis enabled us to determine the average number of prompts and answers for each issue category, as depicted in the bar chart illustrated in Figure. 7.

In the bar chart of Figure. 7, the issue categories are represented on the horizontal axis, while the vertical axis represents the average number of exchanges per conversation. A first quick glance at the graph reveals that *Theoretical Question* has the highest bar, indicating an average of 5.25 exchanges between developers and ChatGPT before reaching a conclusion in a typical conversation. This analysis suggests that *Theoretical Question* tends to be more complex and requires more detailed discussions before a conclusion is reached. Following closely, the *Other* and *Bug* categories are ranked second and third, respectively. Notably, *Feature Request* related conversations are normally shorter as they are concluded within just one or two turns approximately.

*3) Average number of tokens per exchange in a conversation per issue category:*

In our conclusive analysis, our objective is to grasp the typical length of conversations between developers and ChatGPT. In essence, we calculate the average number of tokens in each prompt and answer within a conversation. For the purpose of this analysis, a token is simply defined as a word in a sentence. Once again, this analysis is technically conducted using a dictionary that categorizes all snapshots' conversations into the four defined issue categories. Therefore, our focus in this segment is on discovering the average number of words in both prompts and answers within a conversation. The outcomes of this analysis are visually represented by the means of a bar chart in Figure. 8.

Firstly, an immediate observation is about the dominance

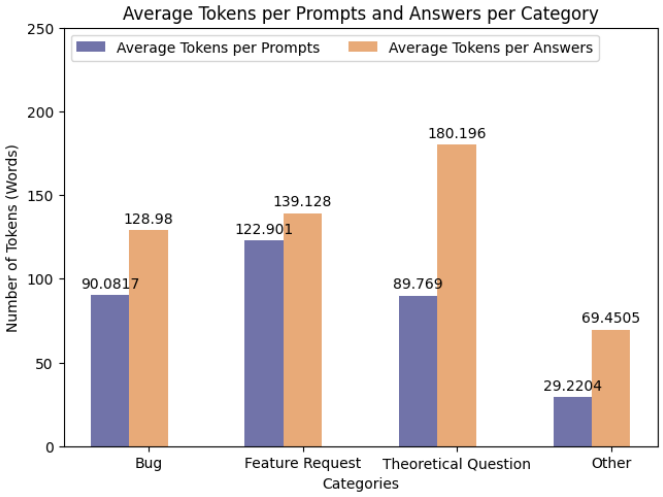


Fig. 8. Results of average tokens per prompts and answer per conversation for each issue category

of *Theoretical Question* among the four categories for average tokens per answer in a conversation, with approximately 180 tokens in an answer. This statistic suggests that ChatGPT's responses to *Theoretical Question* are more elaborated, defined and detailed compared to other issue categories. Interestingly, despite having shorter prompts than *Feature Request*, *Theoretical Question* result in the longest overall conversations due to the lengthy and extensive answers provided by ChatGPT, involving in-depth explanation and exploration of concepts.

Secondly, a notable finding is that *Feature Request* related prompts tend to have the longest average developer prompts, measuring at 122.9 words per prompt. Thus, this analysis implies that conversations in this category may necessitate more detailed input from developers, contributing to longer and more intricate interactions.

Thirdly, the *Other* issue category stands out for its direct and concise nature, as it features shorter prompts and answers on average. This statistic potentially indicates more straightforward resolved issues within this category.

## V. Threats to Validity

We identified two main threats to validity in our work.

Initially, the limited size of the dataset used to train our machine learning model had a direct impact on its precision and the accuracy of our issue categorization results. Therefore, there is significant potential to improve the model's overall performance by enriching and expanding the CSV file with more data. This enhancement will not only refine the model's precision but also provide a more robust and reliable categorization of issues.

Another potential issue is the risk of sampling bias in our study. The training dataset, generated from ChatGPT itself, may not fully represent the diverse range of developer interactions typically encountered. To address this, it's crucial to diversify the dataset. This can be achieved by including prompts from a wide array of project types and domains. Additionally, it's important to consider interactions from developers of varying experience levels, including junior and senior developers, among others. Incorporating data from different geographical regions would also greatly enhance the dataset's diversity, ensuring a more comprehensive and representative sample for training our model.

## VI. Challenges Encountered

During the course of our research study, we encountered two distinct challenges that significantly defined and influenced our process.

Firstly, our initial inexperience with artificial intelligence (AI) and machine learning (ML) posed a significant challenge, particularly in creating a machine learning model from scratch. In the context of software-related academic research, as observed throughout this course, integrating a ML model into a study is often essential to enhance the credibility of the research findings. Despite our limited background in this area, our instinct was to take the risk and follow the same path to answer research question 1. Thus, we were motivated to take on this challenge to address our first research question. This endeavor presented an ideal opportunity for us to engage in research and grasp the basic principles of ML model implementation. Despite encountering numerous obstacles, we ultimately succeeded in developing a model capable of categorizing conversations into various issue types.

Secondly, after realizing the nature of the problem we were addressing, i.e., categorization of conversations into distinct categories, we immediately recognized the importance of visually representing our results through graphs for enhanced visualization and comparison. However, we soon became aware that the feature of representing such visualizations might not be the most convenient in the terminal of an Integrated Development Environment (IDE). Consequently, we opted to transition all of our code to Jupyter Notebook, specifically designed for this visualizations and data analysis purposes. This IDE facilitates the structuring of work into organized code blocks supported by markdown text. The primary challenge in this step involved understanding and familiarizing ourselves with Jupyter Notebook. A difficulty was also associated with the configuration of this tool locally for the desired operation. Despite the initial learning curve, we successfully adapted to the environment. Our Jupyter Notebook now encapsulates all aspects of our study such as blocks of code implementations in Python and results in the form of graphs. We also ensured ourselves to make our Jupyter Notebook really organized while offering a user-friendly and visually intuitive representation of our research.

## VII. Future Works

In our future work, we aim to delve deeper into the categorization of conversations, particularly focusing on those classified under the *Other* section. This category, being a catch-all for various types of exchanges not fitting neatly into predefined categories, presents a rich opportunity for further exploration and refinement.

Additionally, a significant aspect of our future work will involve enhancing the accuracy of our model by incorporating a more diverse and extensive range of data into the training set. The current model, while effective, is based on a dataset that may not fully represent the wide array of possible queries and responses encountered in real-world scenarios. By expanding the dataset to include a broader spectrum of conversational topics, linguistic styles, and complex query types, we can train the model to be more robust and adaptable. This will involve sourcing data from varied demographics, different domains of knowledge, and multiple usage contexts to ensure that the model can accurately categorize an extensive range of interactions.

## VIII. Conclusion

In summary, for research question 1, we employed two distinct implementations to identify the types of issues, i.e., *Bug*, *Feature Request*, *Theoretical Question* or *Other*, that developers normally present to ChatGPT. The *Simple Issue Categorization* proved to be less accurate, with a simplistic implementation which lacked lots of criteria. This lead to a predominant categorization of conversations into the *Other* category. Conversely, the findings from our machine learning approach were much more robust and reliable. Our model, based on TF-IDF and Multinomial Naive Bayes, achieved a 73% of accuracy and precision which improves our results. Thus, we observe that most conversations presented to ChatGPT by developers are usually *Theoretical Question*.

Surprisingly, the results from research question 2, focusing on statistics and the general structure of conversations between developers and ChatGPT, also align with the notion that *Theoretical Question* are the most frequent type of interactions. These conversations exhibit the highest average number of exchanges in a conversation and also include the most tokens on average in a ChatGPT answer.

Finally, it is noteworthy that we did not come across related works on this subject during our research. Our approach was to independently comprehend the problem between us and then collaboratively address and solve it based on our own instincts and ideas as a team. We also extend a special mention to our teacher and teaching assistant who both

provided guidance and validated our methodology as well as our findings.

## REFERENCES


[1] Our GitHub repository for the Final project. https://github.com/keshavDooleea/LOG6307E_Final_Project.

[2] IBM. (2023). *What is Machine Learning?*. IBM. https://www.ibm.com/topics/machine-learning.

[3] IBM. (2023). *What is natural language processing (NLP)?*. IBM. https://www.ibm.com/topics/natural-language-processing.

[4] Ratz, A. V. (2022, April 8). *Multinomial Naive Bayes' For Documents Classification and Natural Language Processing (NLP)*. Medium. https://towardsdatascience.com/multinomial-na%C3%AFve-bayes-for-documents-classification-and-natural-language-processing-nlp-e08cc848ce6.

[5] PhD, E. G. (2023, November 11). *Understanding Multinomial Naive Bayes Classifier*. Medium. https://medium.com/@evertongomede/understanding-multinomial-naive-bayes-classifier-fdbd41b405bf.

[6] LearnDataSci. (n.d.). *TF-IDF — Term Frequency-Inverse Document Frequency*. LearnDataSci. https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/#: :text=Using%20scikit%2Dlearn-.

[7] Kelley, K. (2023, August 4). *What is Data Analysis? Process, Methods, and Types Explained*. Simplilearn. https://www.simplilearn.com/data-analysis-methods-process-types-article.

[8] Levity. (n.d.). *Text Classification: What It Is & How to Get Started*. Levity. https://levity.ai/blog/text-classification.

[9] MonkeyLearn. (2018, October 4). *Text Classification: What it is And Why it Matters*. MonkeyLearn. https://monkeylearn.com/text-classification/.

[10] Databricks. (n.d.). *What is a Jupyter Notebook?*. Databricks. https://www.databricks.com/glossary/jupyter-notebook.