Analysis of Developer Issue Types Presented to ChatGPT

Group 8: Huan He, Jingran Zhao, Yuzhou Wang

1. Introduction

This study aims to explore how developers interact with ChatGPT by analyzing their prompts and the issues they encounter. Specifically, we seek to tackle the following research questions (RQs):

RQ1: Identify the most common types of queries developers present, categorizing them into Bug Reports, Feature Requests, Theoretical Questions, and Other;

RQ2: Determine the average number of conversational turns required to reach a resolution, providing insights into ChatGPT's efficiency in addressing developer needs;

RQ3: Examine common quality issues in SQL code generated by ChatGPT, focusing on errors like "table does not exist," syntax issues, and unrecognized tokens.

By addressing these research questions, our study will contribute to improving the effectiveness and reliability of AI-driven developer support systems.

2. Methodology

2.1 Data Processing

Before conducting further research and analysis, we first converted and merged all available JSON files in folder 'snapshot_20230831' into a structured dataset. The data processing workflow followed key steps, including extracting ChatGPT conversations, processing individual JSON files, and merging all processed data into a unified dataset.

Each JSON file contained conversation data embedded within the ChatgptSharing field, from which key details such as user prompts, ChatGPT responses, code snippets (if present), model type, token usage, conversation date, and share URLs were extracted. Additionally, metadata—including source type (e.g., discussion, commit, file sharing), source URL, author information, title, body content, and timestamp—was retained to preserve context. ChatGPT conversation details were merged with source metadata to ensure linkage to the original context.

Once individual files were processed, all extracted conversations were combined into a single dataset, with duplicates removed based on key attributes such as source type, URL, and number of prompts. Finally, the consolidated dataset was saved as a CSV file for easy access and further analysis. By structuring extracted conversations and metadata into a unified dataset, we ensured that all available data was systematically organized and ready for subsequent research.

2.2 Methodology for RQ1

Data Cleaning:

- Handled missing values in the Prompt column by filling them with empty strings.
- Standardized the text to lowercase for consistent processing.

Issue Classification:

- Developed a rule-based classifier using keyword matching to assign prompts to categories.
- Prompts were classified into four categories: Bug Report, Feature Request, Theoretical Question, and Other.

Visualization:

- The distribution of issue types was calculated as percentages and visualized using a bar chart.

2.3 Methodology for RQ2

Data Cleaning:

- Removed duplicate records based on the combination of 'SourceType', 'SourceURL', and 'NumberOfPrompts'.

Grouping and Calculation:

- Grouped by the 'SourceType' column to calculate the mean of 'NumberOfPrompts' for each type.
- Sorted in descending order of the average number of prompts to highlight source types that typically require more interactions to reach a conclusion.

Visualization:

- The average number of prompts for each source type was calculated and visualized using a bar chart.

2.4 Methodology for RQ3

Data Cleaning:

- Handled missing or invalid data in the "ListOfCode" column by filtering out empty or invalid strings.
- Parsed the 'ListOfCode' strings into Python dictionaries using ast.literal_eval to facilitate extraction of relevant information.

Code Type Classification:

- Analyzed the "ListOfCode" column to identify and categorize the top 10 most common code types present in the dataset.

- A bar chart was created to visualize the distribution of these top 10 code types, providing insight into the predominant code categories in the dataset.

Error Classification:

- Developed a function to simulate the execution of each SQL snippet using an in-memory SQLite database.
- Each SQL code was evaluated, and errors were categorized into types such as "table does not exist", "Syntax error", or "unrecognized token".
- The error messages were mapped to corresponding SQL code snippets to track error frequencies.

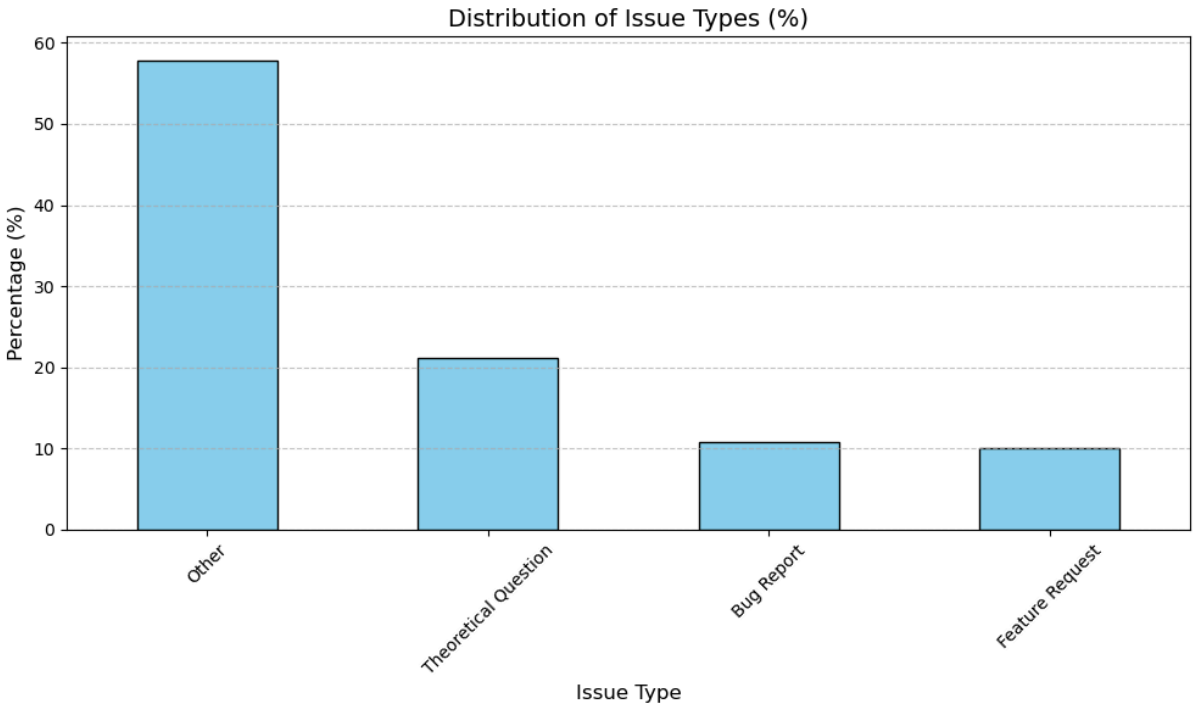
Visualization:

- Top 10 Most Common Code Types: A bar chart displaying the most frequent code types in the dataset.
- SQL Error Type Distribution: A horizontal bar chart showing the distribution of SQL error types, emphasizing the most common issues developers face.

3. Results

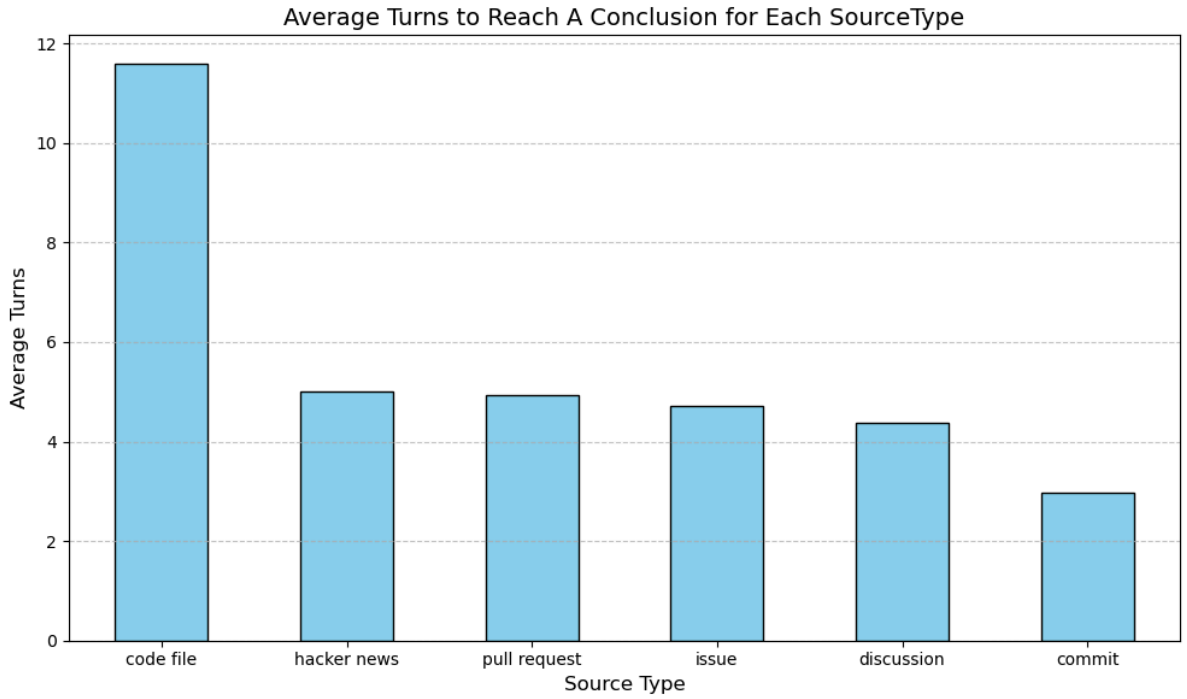
3.1 Results for RQ1

- The majority of prompts fell into the Other category (~60%), indicating diverse or unstructured queries.
- Theoretical Questions accounted for ~20%, reflecting a significant use of ChatGPT for conceptual assistance.
- Bug Reports and Feature Requests were less common (~10% each).



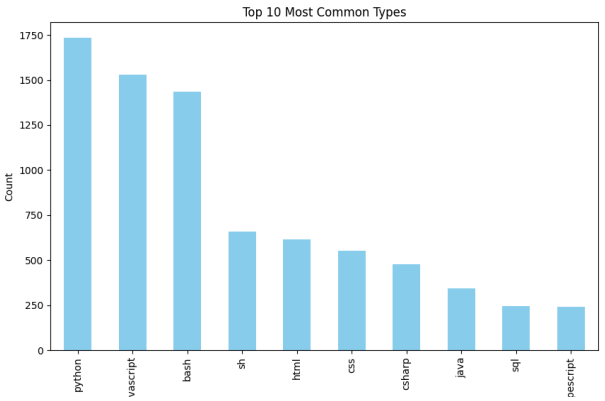
3.2 Results for RQ2

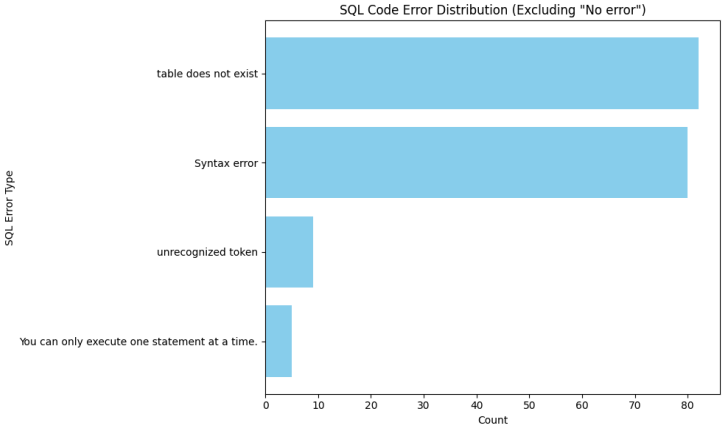
- Code files require the highest number of turns (11.59) to reach a solution, suggesting that resolving or understanding issues related to code files is more complex and time-consuming.
- Hacker News, pull requests, issues, and discussions fall within a similar range, with average turns ranging from 4.37 to 5.02.
- Commits have the lowest average number of turns (2.98), indicating that conclusions related to commits are reached more quickly.



3.3 Results for RQ3

- Python, JavaScript, and Bash code types made up a significant portion of the dataset, with around 1,500 entries. These languages were among the most prevalent in the "ListOfCode" field, indicating frequent usage by developers.
- The most common errors encountered during the simulation were "table does not exist", followed by "Syntax error".
- A smaller proportion of errors were "unrecognized token" issues.





4. Interpretation

4.1 Interpretation for RQ1

Developers frequently use ChatGPT for unstructured queries or general guidance, as reflected in the large Other category, while a significant portion of queries seek theoretical help, highlighting the importance of strong explanatory capabilities. In contrast, Bug Reports and Feature Requests appear less common, suggesting that these queries are more niche or specific.

These insights imply that conversational AI systems like ChatGPT could benefit from further optimization for theoretical discussions and broad guidance. Additionally, future improvements should focus on better understanding and addressing the diverse and less-defined queries within the Other category to enhance overall usability and effectiveness.

4.2 Interpretation for RQ2

There is significant variation in the number of conversational turns required to reach a conclusion across different source types. Code files tend to require more turns, likely due to their technical nature, which demands detailed analysis and multiple iterations. In contrast, commits generally require fewer turns, as they represent discrete changes that are easier to review. Hacker News, pull requests, issues, and discussions involve interactive and collaborative elements, requiring some back-and-forth but generally less complexity than code files.

These insights suggest that teams should allocate more time and resources for tasks involving code reviews and debugging, potentially through dedicated review sessions or automated analysis tools. Understanding the average number of turns for different source types can aid in task prioritization, with quick resolutions for commits and additional buffer time for complex code reviews. For mid-range sources like pull requests, issues, and discussions, optimizing collaboration through clear documentation, structured communication, and defined workflows can improve efficiency and keep discussions focused.

4.3 Interpretation for RQ3

The prevalence of "table does not exist" errors suggests that developers frequently struggle with missing or misnamed tables in their SQL queries, while "syntax error" indicates issues with SQL structure or improper syntax. The relatively low occurrence of "unrecognized token" errors implies fewer problems with invalid characters or unknown keywords. Additionally, the "you can only execute one statement at a time" error highlights challenges with running multiple queries in a single execution context, leading to conflicts.

These insights suggest that improving AI systems like ChatGPT to generate more accurate SQL queries and provide better error handling could greatly benefit

developers. Future enhancements should focus on assisting users in resolving these common errors more effectively, ensuring smoother SQL query execution and debugging.

5. Conclusion

This study systematically processed and explored the provided data and revealed key insights into how developers interact with ChatGPT, highlighting areas for improvement in AI-assisted development.

Regarding RQ1, we identified the primary types of queries developers present, with a significant portion falling under unstructured or theoretical guidance, suggesting opportunities for deeper semantic classification and predictive issue-type modeling.

RQ3 revealed notable variations in the number of turns required to reach a conclusion across different source types, with code files requiring the most iterations due to their complexity, commits resolving more quickly, and pull requests, issues, and discussions falling in between. These findings underscore the need for tailored approaches to different tasks, such as allocating more resources for complex code reviews and streamlining collaborative processes.

RQ6 uncovered common quality issues in SQL code generated by ChatGPT, including frequent errors like "table does not exist," "syntax error," and "you can only execute one statement at a time." Addressing these recurring problems can enhance ChatGPT’s ability to generate more accurate SQL code and assist developers in debugging more efficiently.

Together, these findings provide a foundation for optimizing AI-driven developer support and improving ChatGPT’s effectiveness in handling diverse queries.