Benchmark Studio

LLM MODEL PERFORMANCE COMPARISON ANALYSIS

Report Milestone 2

Report by:

Akshara Sri Lakshmipathy - akla8196@colorado.edu

Harish Nandhan Shanmugam - hash1366@colorado.edu

Shivaraj Senthil Rajan - shse1502@colorado.edu

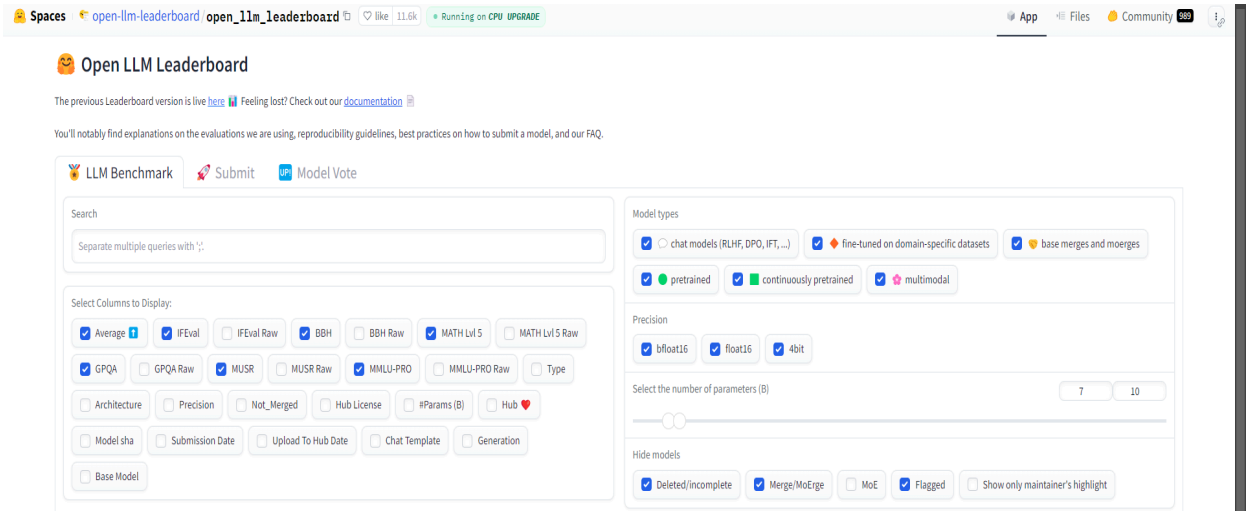
Abstract:

Currently, Large Language Models constitute the center of Generative Artificial Intelligence, which is a fast-growing area. These models understand text, voice, or image prompts and generate content with each prompt accordingly. The project aims to find the best LLM for a specific task's performance by analyzing the benchmark results for various LLMs. This will mainly focus on the factors and how each contributes to the performance of LLMs using exploratory data analysis. This would be done using publicly available data from various research papers, model repositories, and benchmark reports. Quite a few AI models have been developed in this era of AI, especially by giants like Google, OpenAI, Amazon, and Meta. Choosing the best among them has hence been an exhausting task. This analysis will minimize the time a user has to spend selecting a model for themselves; instead, it will get them the best pick that suits the work at hand. It will also predict which model will last longer and give better responses, considering past evidence and other variables. It is expected that from this analysis, valuable insight will be gained into the efficiency and scalability of various LLM architectures, hence contributing to developing more efficient and resource-effective models. Comparisons between open-source and proprietary models will also be explored.

Data Collection and Data Mining:

In this project we have fetched the latest battle data from the external source using the `requests` library. We first saved the JSON to a local file by the name `local_file_name.json`. Then, we loaded this JSON into a pandas DataFrame and sorted it according to the timestamp, `tstamp`, in ascending order. After sorting, we saved the DataFrame as a CSV file named `battles_data.csv`. The verification at the end demonstrates the data has been saved to the necessary file for further analysis.

Datasource:



Data Cleaning:

Importing the Mined Data using Pandas:

	model_a	model_b	winner	judge	turn	anony	language	tstamp	conv_metadata	is_code	is_refusal	dedup_tag	category_tag
0	chatglm-6b	koala-13b	model_b	2e9c29aa140b8e50643235eab01dc9ea	1	True	English	1.682352e+09	{'sum_user_tokens': 10, 'sum_assistant_a_token...	True	False	{'high_freq': False, 'sampled': True}	{'if_v0.1': {'if': False, 'score': 1}, 'math_v...
1	oasst-pythia-12b	alpaca-13b	tie	2e9c29aa140b8e50643235eab01dc9ea	1	True	English	1.682352e+09	{'sum_user_tokens': 11, 'sum_assistant_a_token...	False	False	{'high_freq': False, 'sampled': True}	{'if_v0.1': {'if': False, 'score': 0}, 'math_v...
2	koala-13b	oasst-pythia-12b	model_b	2e9c29aa140b8e50643235eab01dc9ea	1	True	English	1.682352e+09	{'sum_user_tokens': 10, 'sum_assistant_a_token...	False	False	{'high_freq': False, 'sampled': True}	{'if_v0.1': {'if': False, 'score': 0}, 'math_v...
3	vicuna-13b	oasst-pythia-12b	model_b	2e9c29aa140b8e50643235eab01dc9ea	1	True	English	1.682352e+09	{'sum_user_tokens': 9, 'sum_assistant_a_tokens...	False	False	{'high_freq': False, 'sampled': True}	{'if_v0.1': {'if': False, 'score': 1}, 'math_v...
4	vicuna-13b	koala-13b	model_a	2e9c29aa140b8e50643235eab01dc9ea	1	True	English	1.682352e+09	{'sum_user_tokens': 5, 'sum_assistant_a_tokens...	False	True	{'high_freq': False, 'sampled': True}	{'if_v0.1': {'if': False, 'score': 0}, 'math_v...

We are converting Unix timestamps in seconds into a more readable date format (YYYY-MM-DD). For instance 1609459200 this will represent in this format 2021-01-01 and further we convert the 'tstamp' column to a proper datetime format using pd.to_datetime. Then, we create a new column, called 'month_year', which extracts the month and year from the timestamp and stores it as a period, say '2024-08'. Here, we will extract the unique dates in the 'tstamp' column filtered DataFrame by changing the datetime values to date format. We then convert these unique dates into a list and display the list of unique dates for further analysis.The next step where we exclude the specific date from a DataFrame, and filter that DataFrame to remove any rows that contain these excluded dates. The certain dates that are August 1, 2, 3 in 2024 are omitted from analysis while the data includes only the time period between Aug 4 to 14 in 2024.

	tstamp
0	2023-04-24
1	2023-04-24
2	2023-04-24
3	2023-04-24
4	2023-04-24

```
[Period('2023-04', 'M'), Period('2023-05', 'M'), Period('2023-06', 'M'), Period('2023-07', 'M'), Period('2023-08', 'M'), Period('2023-09', 'M'), Period('2023-10', 'M')
```

tstamp
2024-08-01
2024-08-01
2024-08-01
2024-08-01
2024-08-01

And also we removed the columns judge and anony from the dataset because columns judge and anony unnecessary columns so we preprocessed to clean up the DataFrame and stored in filtered_excluded_df

	model_a	model_b	winner	turn	language	tstamp	conv_metadata	is_code	is_refusal	dedup_tag	category_tag	month_year
1725966	claude-3-opus-20240229	gemma-2-2b-it	model_b	1	French	2024-08-04	{'sum_user_tokens': 19, 'sum_assistant_a_token...	False	False	{'high_freq': False, 'sampled': True}	{'if_v0.1': {'if': True, 'score': 4}, 'math_v0...	2024-08
1725967	reka-core-20240722	athene-70b-0725	model_b	3	English	2024-08-04	{'sum_user_tokens': 51, 'sum_assistant_a_token...	False	False	{'high_freq': False, 'sampled': True}	{'if_v0.1': {'if': True, 'score': 4}, 'math_v0...	2024-08
1725968	athene-70b-0725	llama-3.1-70b-instruct	model_a	6	Italian	2024-08-04	{'sum_user_tokens': 472, 'sum_assistant_a_toke...	False	False	{'high_freq': False, 'sampled': True}	{'if_v0.1': {'if': False, 'score': 0}, 'math_v...	2024-08
1725969	llama-3-70b-instruct	llama-3.1-8b-instruct	model_a	2	English	2024-08-04	{'sum_user_tokens': 84, 'sum_assistant_a_token...	False	False	{'high_freq': False, 'sampled': True}	{'if_v0.1': {'if': True, 'score': 4}, 'math_v0...	2024-08
1725970	llama-3.1-70b-instruct	gpt-4-turbo-2024-04-09	model_b	1	French	2024-08-04	{'sum_user_tokens': 29, 'sum_assistant_a_token...	False	False	{'high_freq': False, 'sampled': True}	{'if_v0.1': {'if': False, 'score': 1}, 'math_v...	2024-08
...	...	...	...	...	...	...	...	...	...	...	...	...
1799986	llama-3-8b-instruct	chatgpt-4o-latest	model_a	1	English	2024-08-	{'sum_user_tokens': 18,	False	False	{'high_freq': False, 'sampled':	{'if_v0.1': {'if': False, 'score': 1},	2024-08

And for further analysis we created a new column called 'winner_model', which selects either the 'model_a' or 'model_b' depending on who was the winner. Then, it removes the columns 'model_a', 'model_b', and 'winner' because they are not needed once the column 'winner_model' has been created. The cleaned DataFrame is then returned without showing the index.

model
gemma-2-2b-it
athene-70b-0725
athene-70b-0725
llama-3-70b-instruct
gpt-4-turbo-2024-04-09

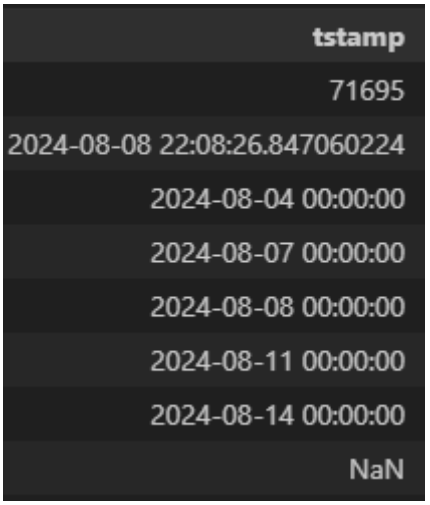
Also we define a function, extract_conv_metadata, that does safe parsing on 'conv_metadata'. Then, it extracts the appropriate token information for the winning model through user tokens, assistant tokens, and context tokens. We use this function to a pply to every row in the DataFrame and create new columns out of the metadata extracted. Finally, we join these new columns with the original DataFrame, drop the original 'conv_metadata' column, and display the updated DataFrame without the index and we filter the DataFrame for those rows for which 'language' does not equal 'unknown'. Then, we display this updated DataFrame without showing an index to make certain that we retain only entries of known languages going forward. We will be able to get the unique models by pulling a list of unique values from the 'model' column in the DataFrame.

sum_user_tokens	sum_assistant_tokens	context_tokens
19	328	19
51	1846	1434
472	3504	3396
84	1047	580
29	556	29

Then we are going to find a length of this list, which is going to give us a total count of unique models. we print both the list of unique models and its length for the user to analyze further. Again we define a function extract_dedup_tag, which safely evaluates the column 'dedup_tag' to extract the values for key 'high_freq' and 'sampled'. Then, we apply this function onto every row of column 'dedup_tag', creating two new columns: 'dedup_tag_high_freq' and 'dedup_tag_sampled'. Later, we do not need 'dedup_tag', and hence we print the updated DataFrame.

dedup_tag_high_freq	dedup_tag_sampled
False	True
False	True
False	True
False	True
False	True

We convert the 'tstamp' column to a period with daily frequency and store it in a new column named 'tstamp_period'. Optionally, one may also drop the original column 'tstamp', if it won't be used anymore. Finally, we print a few lines of the updated DataFrame to verify and converts the column 'category_tag' from a string in JSON-like format to a dictionary format using the ast.literal_eval() function.



tstamp
71695
2024-08-08 22:08:26.847060224
2024-08-04 00:00:00
2024-08-07 00:00:00
2024-08-08 00:00:00
2024-08-11 00:00:00
2024-08-14 00:00:00
NaN

We then extracted some of these values from this dictionary into separate columns, such as 'information_fulfillment', 'Math', 'specificity', and many more. Once the extraction of these values from the dictionary is affected, the original column 'category_tag' can well be dropped and the new DataFrame returned to see the new columns.

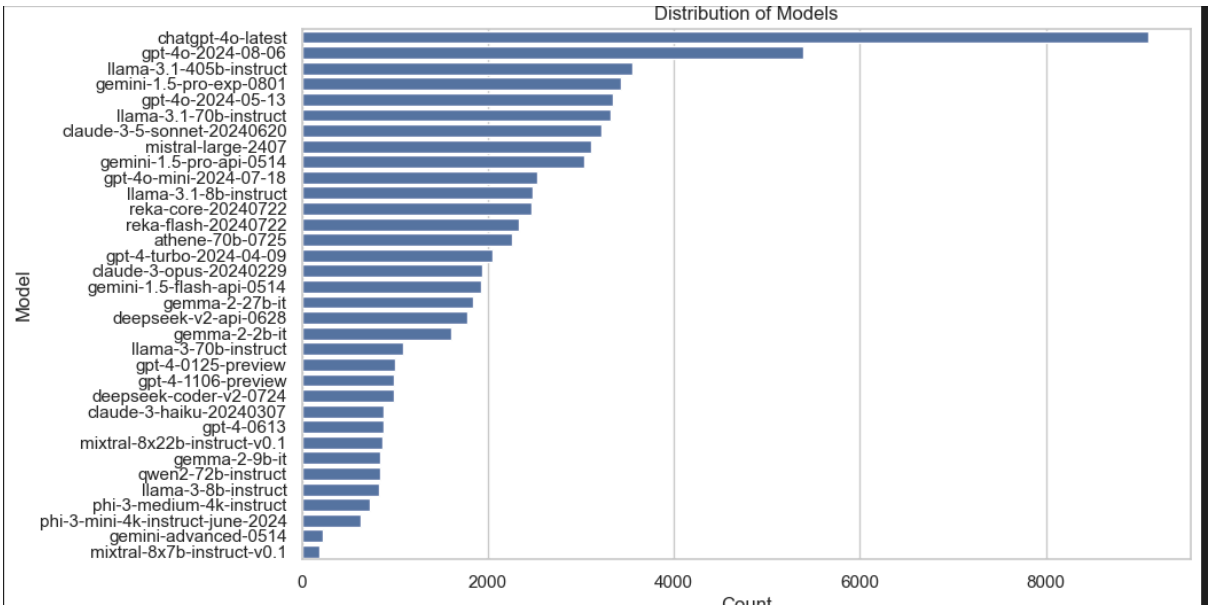
Math	specificity	domain_knowledge	complexity	problem_solving	creativity	technical_accuracy	real_world
False	True	True	False	True	True	False	True
False	True	True	True	True	False	True	True
False	False	True	False	True	True	True	True
False	True	True	False	True	False	True	True
False	True	True	False	False	False	True	True

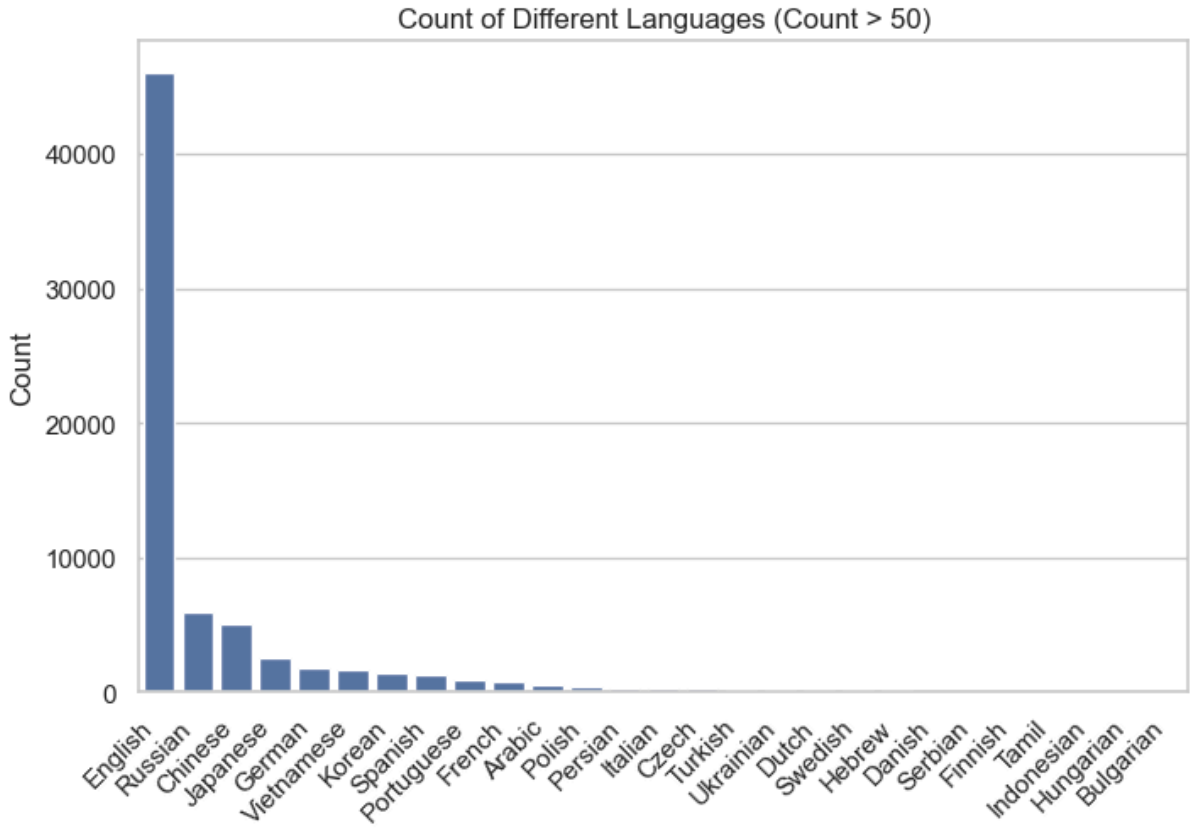
And we came to know that removing the columns category_tag, tstamp, dedup_tag, and month_year from the dataset are unnecessary columns so we can concentrate on the remaining relevant data. This is the final step in which we are going to save our cleaned DataFrame-which is df_cleaned-into a CSV file named "battles_data_cleaned.csv". The index=False just makes sure that the index from the DataFrame will not appear in the file that will be saved, thus keeping things clean for future use.

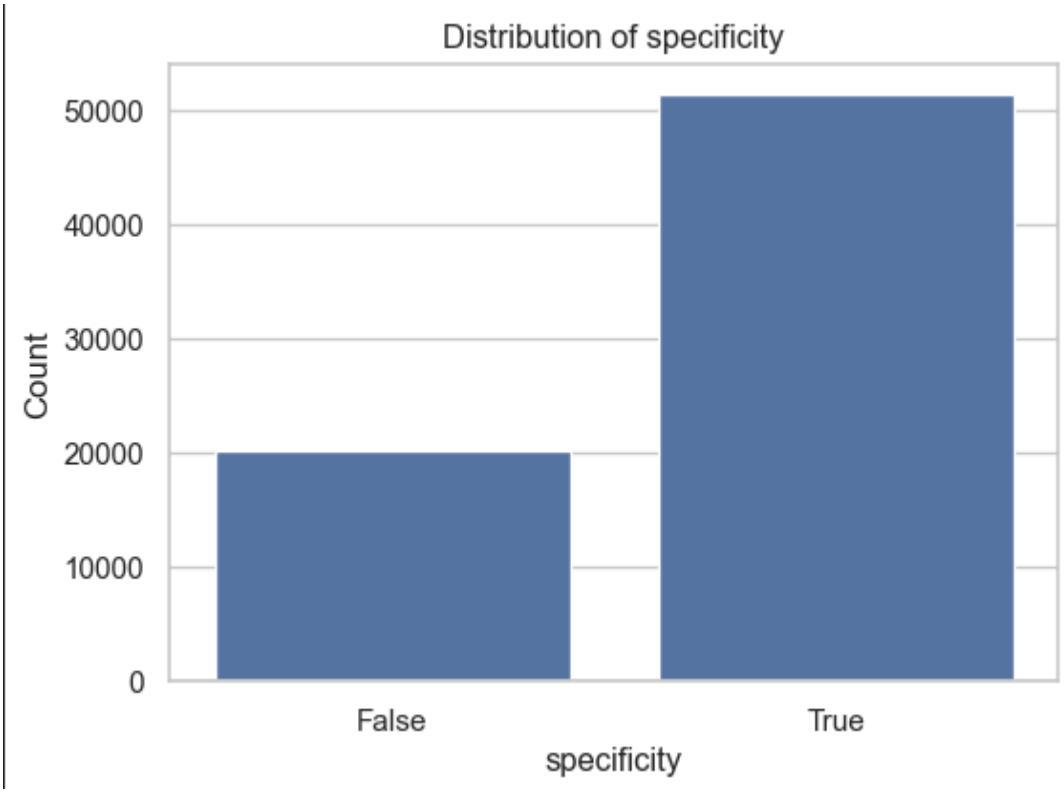
turn	language	is_code	is_refusal	model	sum_user_tokens	sum_assistant_tokens	context_tokens	dedup_tag_high_freq	dedup_tag_sampled	tstamp_period	information_fulfillment	Math	specificity	domain_knowledge	complexity	problem_solving	creativity	technical_accuracy	real_world
1	French	False	False	gemma-2-2b-it	19	328	19	False	True	2024-08-04	True	False	True	True	False	True	True	False	True
3	English	False	False	athene-70b-0725	51	1846	1434	False	True	2024-08-04	True	False	True	True	True	True	False	True	True
6	Italian	False	False	athene-70b-0725	472	3504	3396	False	True	2024-08-04	False	False	False	True	False	True	True	True	True
2	English	False	False	llama-3-70b-instruct	84	1047	580	False	True	2024-08-04	True	False	True	True	False	True	False	True	True
1	French	False	False	gpt-4-turbo-2024-04-09	29	556	29	False	True	2024-08-04	False	False	True	True	False	False	False	True	True

Data Visualization:

Distribution of Models, Count of Diffex`rent Languages with More than 50 Occurrences, Distribution of Boolean Features (is_code, is_refusal, etc.)


Distribution of Models
Model
chatgpt-4o-latest
gpt-4o-2024-08-06
llama-3.1-405b-instruct
gemini-1.5-pro-exp-0801
gpt-4o-2024-05-13
llama-3.1-70b-instruct
claude-3-5-sonnet-20240620
mistral-large-2407
gemini-1.5-pro-api-0514
gpt-4o-mini-2024-07-18
llama-3.1-8b-instruct
reka-core-20240722
reka-flash-20240722
athene-70b-0725
gpt-4-turbo-2024-04-09
claude-3-opus-20240229
gemini-1.5-flash-api-0514
gemma-2-27b-it
deepseek-v2-api-0628
gemma-2-2b-it
llama-3-70b-instruct
gpt-4-0125-preview
gpt-4-1106-preview
deepseek-coder-v2-0724
claude-3-haiku-20240307
gpt-4-0613
mixtral-8x22b-instruct-v0.1
gemma-2-9b-it
qwen2-72b-instruct
llama-3-8b-instruct
phi-3-medium-4k-instruct
phi-3-mini-4k-instruct-june-2024
gemini-advanced-0514
mixtral-8x7b-instruct-v0.1
0
2000
4000
6000
8000
Count


Count of Different Languages (Count > 50)
Count
40000
30000
20000
10000
0
English
Russian
Chinese
Japanese
German
Vietnamese
Korean
Spanish
Portuguese
French
Arabic
Polish
Persian
Italian
Czech
Turkish
Ukrainian
Dutch
Swedish
Hebrew
Danish
Serbian
Finnish
Tamil
Indonesian
Hungarian
Bulgarian

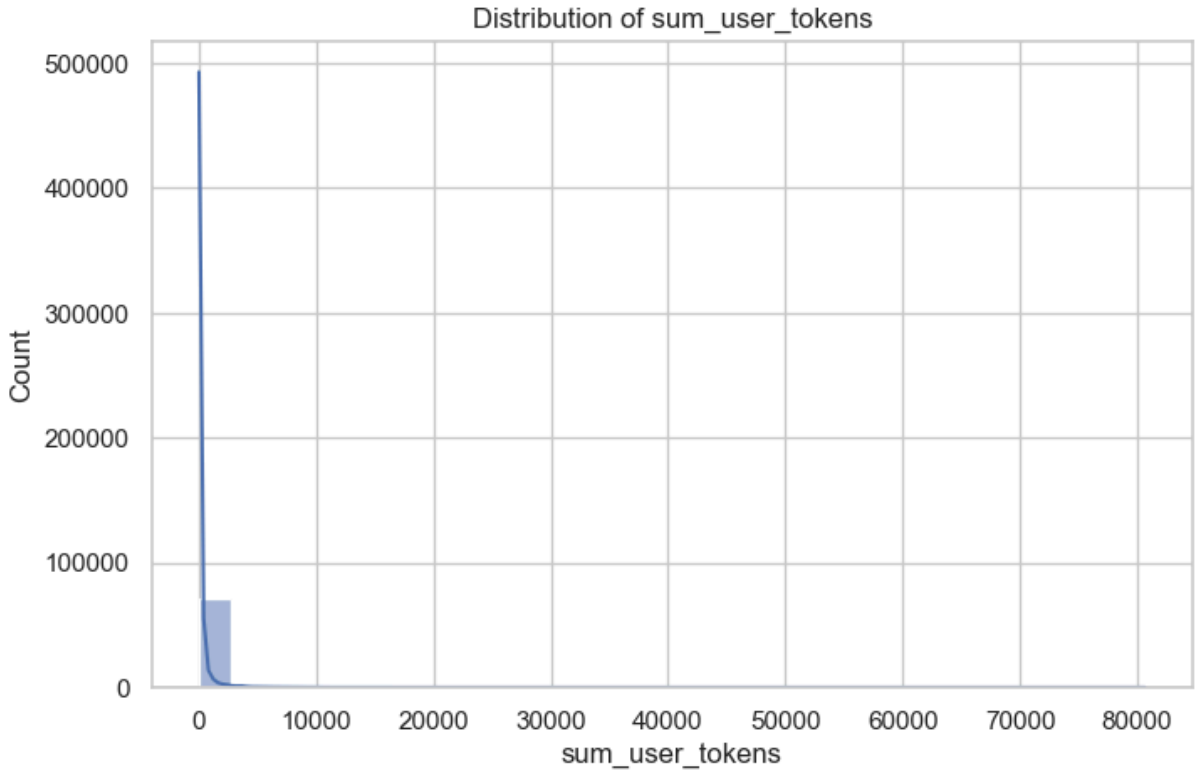


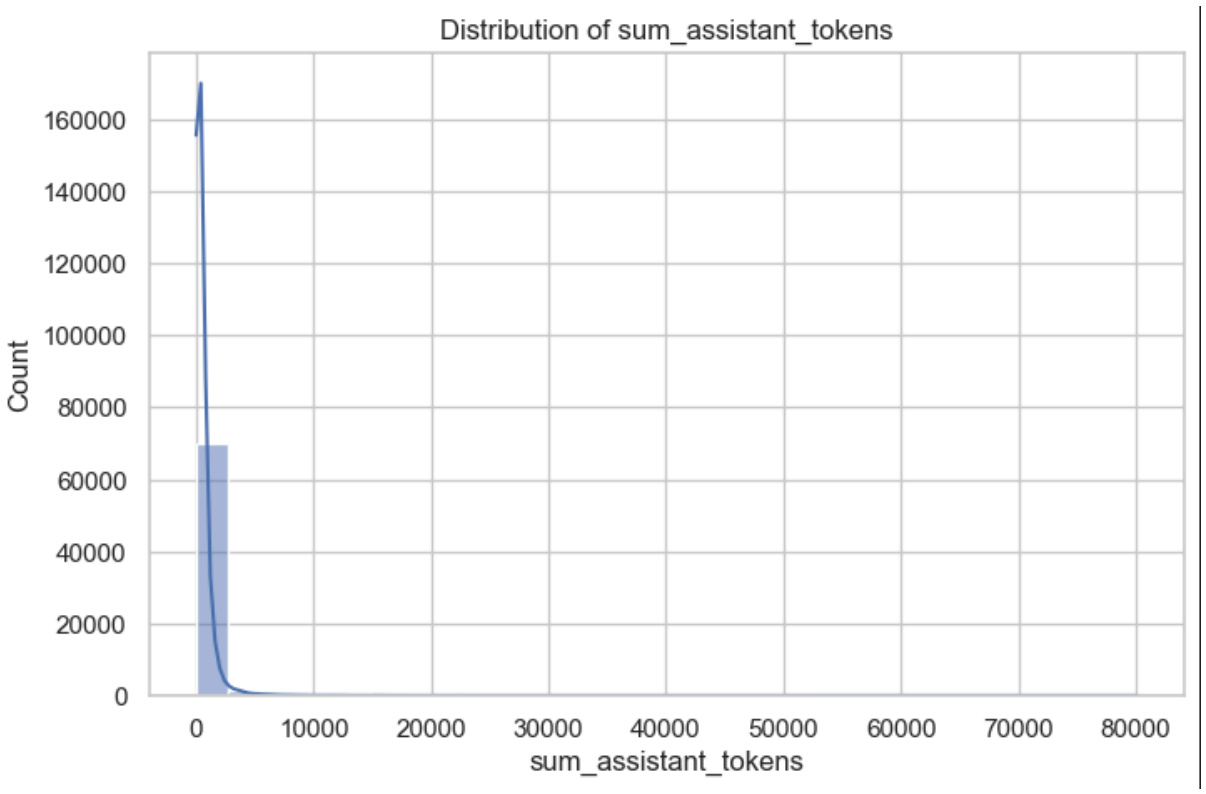
In the plot distribution of models it visualizes which models is most frequent and which is less common. Each bar represents a model, and its length corresponds to the count of occurrences of that model.

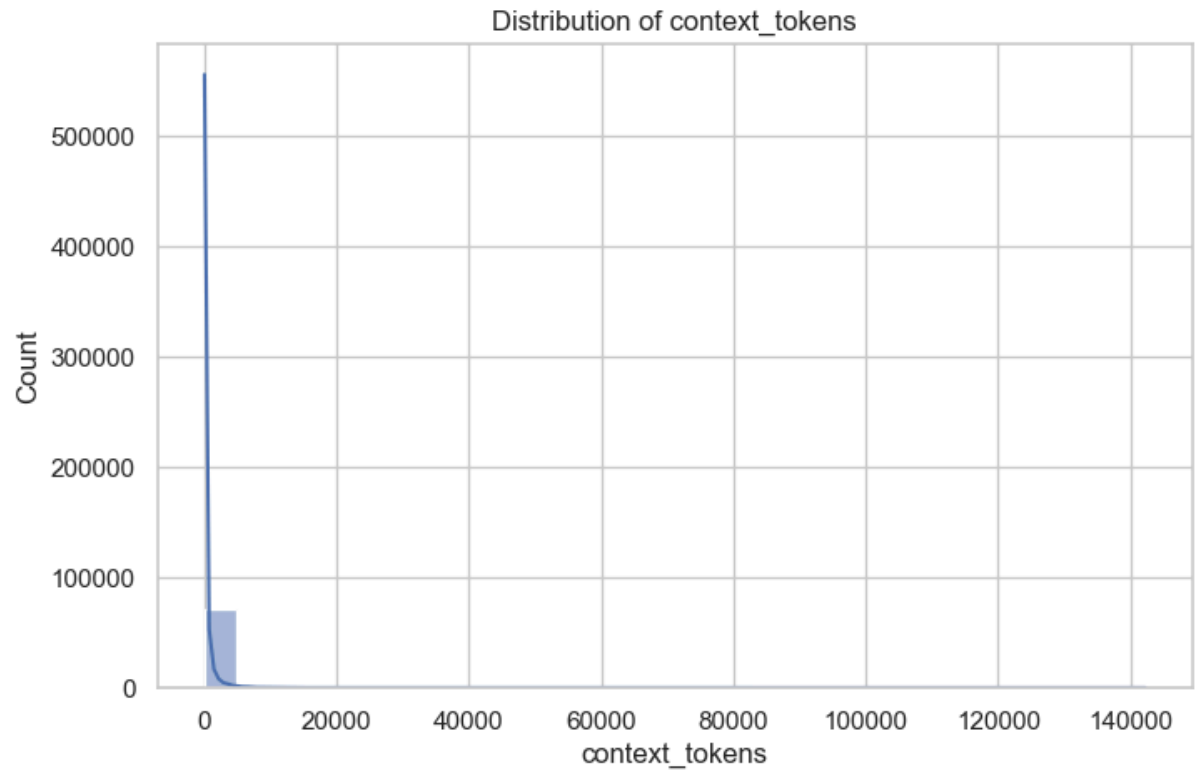
The second plot represents the count of different languages that have more than 50 counts. We are filtering out languages with only more than 50 counts because plots will make easier to focus on the more frequently occurring ones or else the plot will be clumsy.

The rest of the graph explains the Boolean Columns
- is_code: There is higher count in False which indicates that it's not associated with code or programming tasks where it suggesting a focus on non-technical content.
- is_refusal: There is a higher count in False which indicates that input was accepted where users are responsive to input.
- information_fulfillment: There is a higher count in False which indicates that the model information needs were not met.
- math: There is a higher count in False which indicates a lack of mathematical proficiency where suggesting that it does'nt involve mathematical reasoning.
- specificity: There is a higher count in True which indicates that the model information is specific, which provides detailed insights.
- domain_knowledge: There is higher count in True which indicates that the model content is strong in particulars field.
- complexity: There is higher count in False which indicates that the complexity of the topic is addressed which is straightforward means simpler approach to question.
- problem_solving: There is higher count in False which indicates that the No problem-solving is clear, which may indicate there is a lack of critical thinking.
- critical thinking: There is higher count in False which indicates that there is a lack of creativity in the model showed in the response.
- technical_accuracy: There is higher count in True which indicates that model provides the technical information is accurate and reliable.
- real_world: There is higher count in True which indicates that the model content connects with real-world applications.

Distribution of User Tokens, Assistant Tokens and Context Tokens




Distribution of sum_assistant_tokens
Count
160000
140000
120000
100000
80000
60000
40000
20000
0
0
10000
20000
30000
40000
50000
60000
70000
80000
sum_assistant_tokens

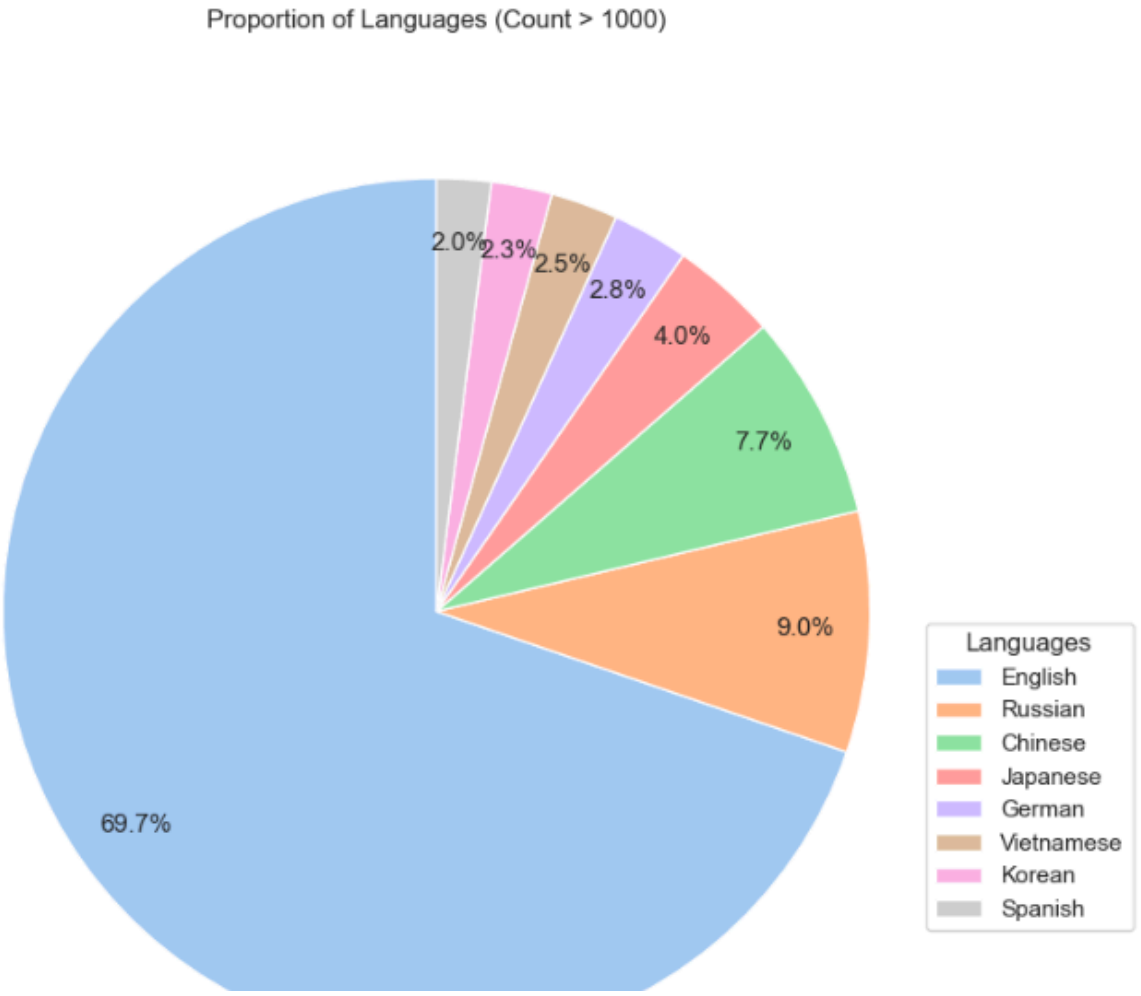


The plot distribution of sum_user_tokens visualizes with the Kernel Density Estimate (KDE) where shows that the majority of the token values are near zero, with a line extending toward higher values 500000. This indicates that most users have relatively low token usage, while a smaller subset has significantly higher usage.

The second plot the distribution of sum_assistant_tokens also visualizes with the Kernel Density Estimate (KDE) where the majority of assistant_tokens are clustered upto 700000 where the majority of assistant responses involve a small number of tokens, there are occasional responses with a much higher token count.

The third plot distribution of context_tokens also visualizes with the Kernel Density Estimate (KDE) where it's similar to the first plot, where the majority of context_tokens values are near zero and a line extending to the 500000 values. The most contexts involve a small number of tokens, with a few contexts containing significantly more which means that in our dataset, the majority of entries have a low number of tokens (e.g., words, characters, or data units in each context), while only a small number of entries have a much higher token count.

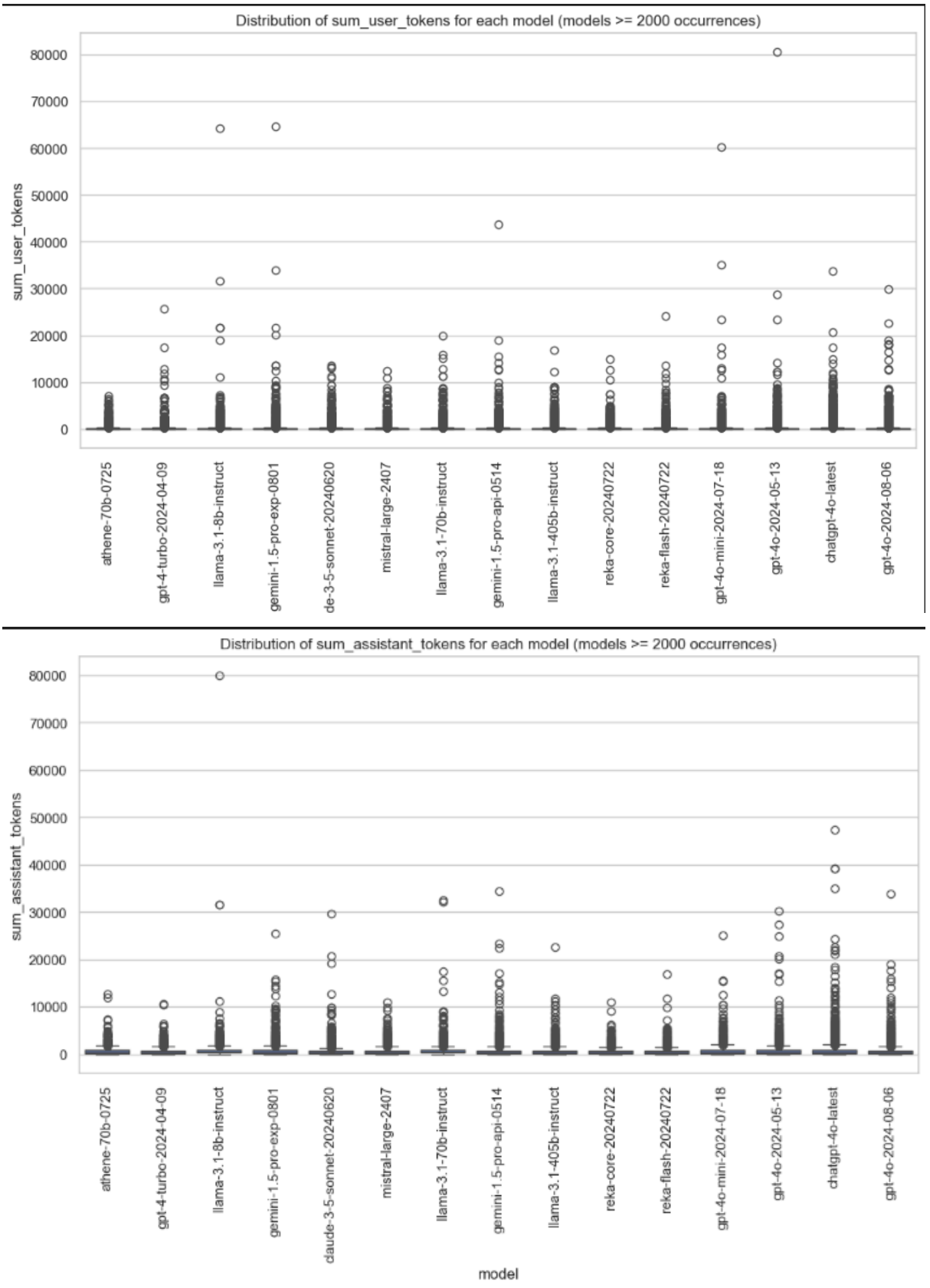
Proportion of Languages with Over 1000 Occurrences



The pie chart below displays the proportion of languages with over 1000 occurrences in the dataset, with each segment being representative of a language. This is indicated with the size of the slice proportional to the count. The dominant proportion is English at almost 70%, followed by Russian, Chinese, and others.

This plot helps in model development by highlighting the language distribution, with developers focusing on optimizing models for prevalent languages. It also helps in model selection since, while dealing with multilingual tasks, sometimes the model choice needs to be made based on support for a particular language.

User and Assistant Token Distributions for Models with Over 2000 Occurrences

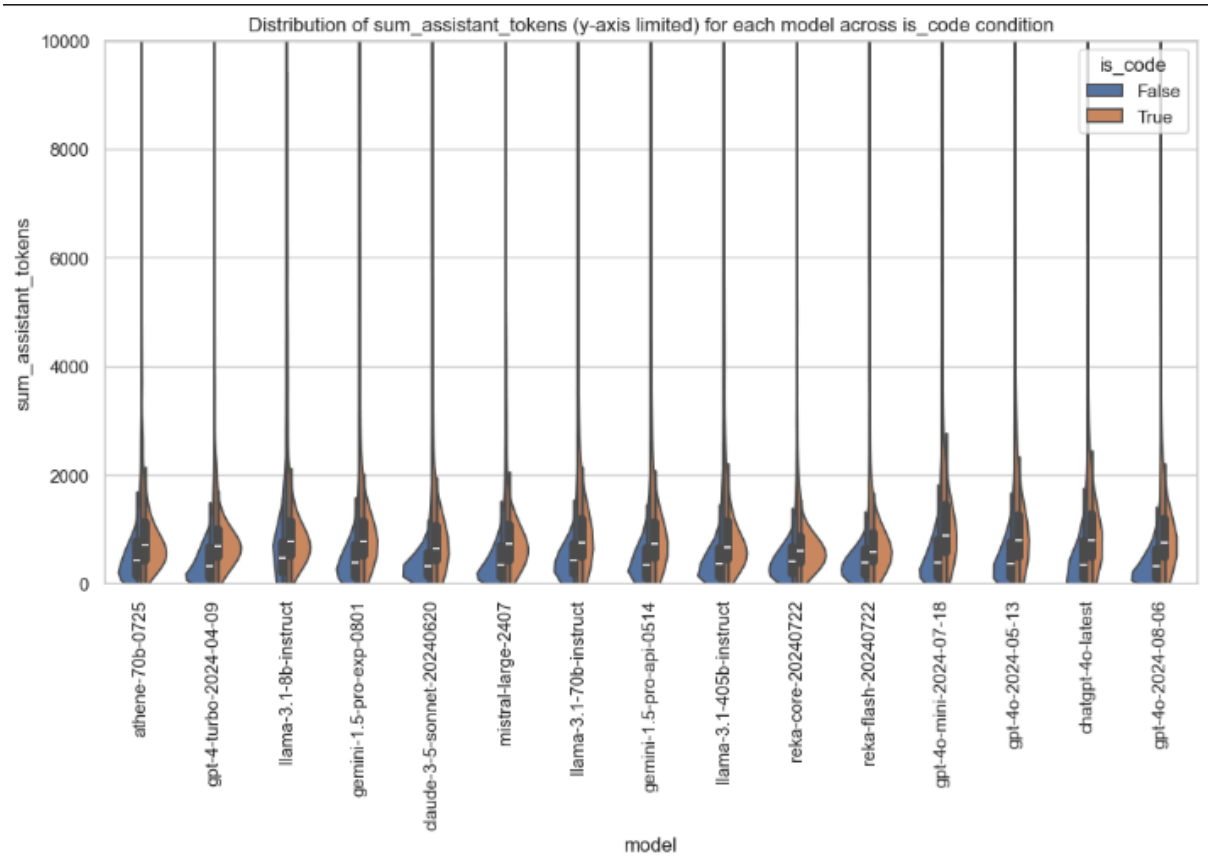


These are box plots of the distribution of sum_user_tokens and sum_assistant_tokens for models with at least 2000 occurrences. These box plots depict the dispersion in token usage

from each model: the median represented by the central line of the box, interquartile range by the size of the box itself, and possible outliers as dots outside the whiskers.

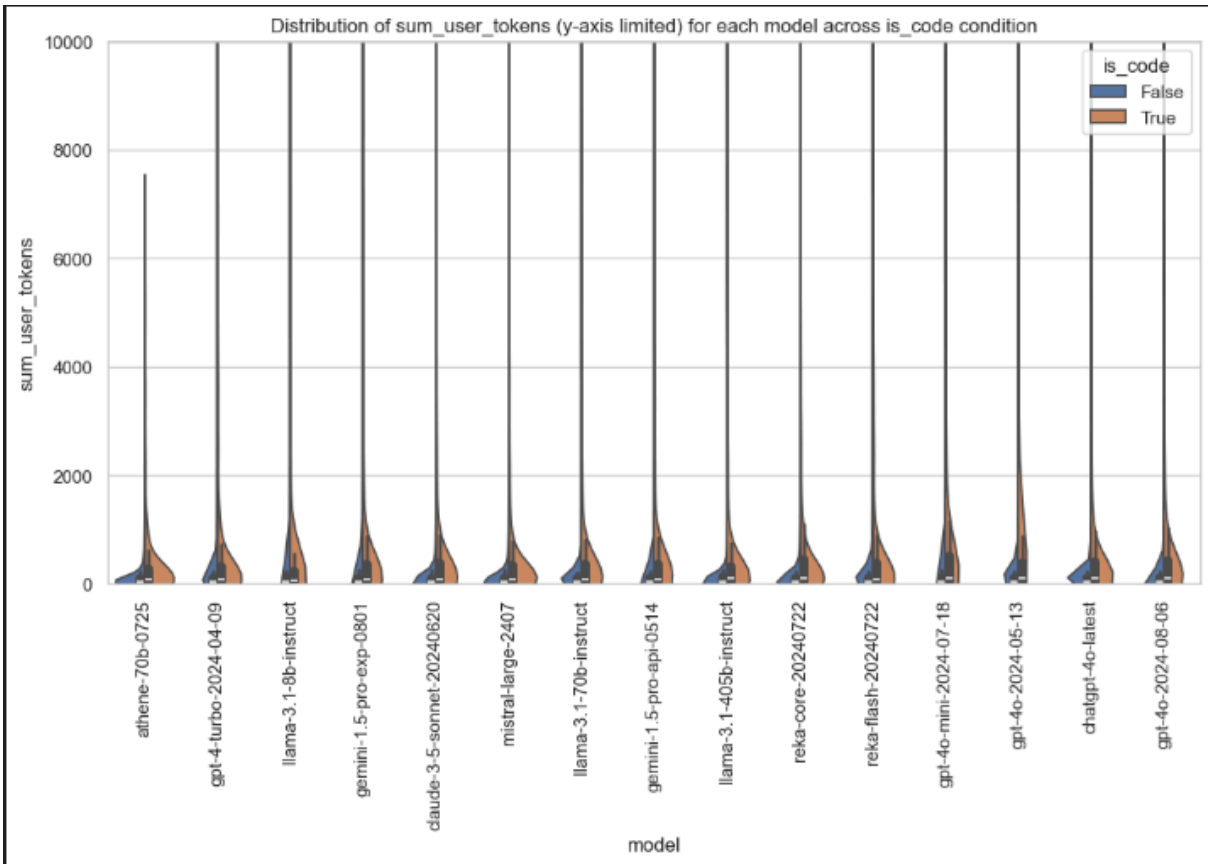
These plots are useful in the model development stage by underlining token usage consistency and outliers that could be used to optimize the handling of the tokens. They also contribute to model selection by comparing different models based on token efficiency.

Assistant Token Distribution Across Models for Code and Non-Code Inputs



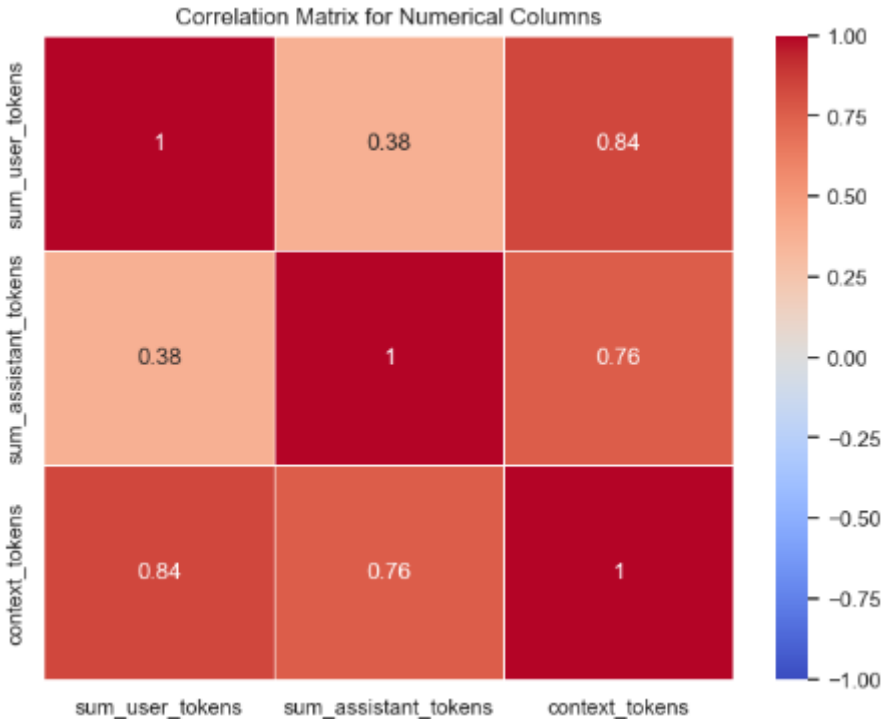
The following violin plot depicts the distribution of sum_user_tokens for each model, segregated based on the condition if is_code is true or false. The y-axis limits are set to a maximum of 10,000 for better clarity. Herein, the violin plot shows the density and range of token usage; hence, one can get insights about how each model is behaving with and without code input. This plot helps in model development for understanding the pattern of token usage when handling code and helps in model selection, pointing out those models which manage efficiently with token heavy tasks, more so for code.

User Token Distribution Across Models for Code and Non-Code Inputs



The following violin plot depicts the distribution of sum_user_tokens for each model, segregated based on the condition if is_code is true or false. The y-axis limits are set to a maximum of 10,000 for better clarity. Herein, the violin plot shows the density and range of token usage; hence, one can get insights about how each model is behaving with and without code input. This plot helps in model development for understanding the pattern of token usage when handling code and helps in model selection, pointing out those models which manage efficiently with token heavy tasks, more so for code.

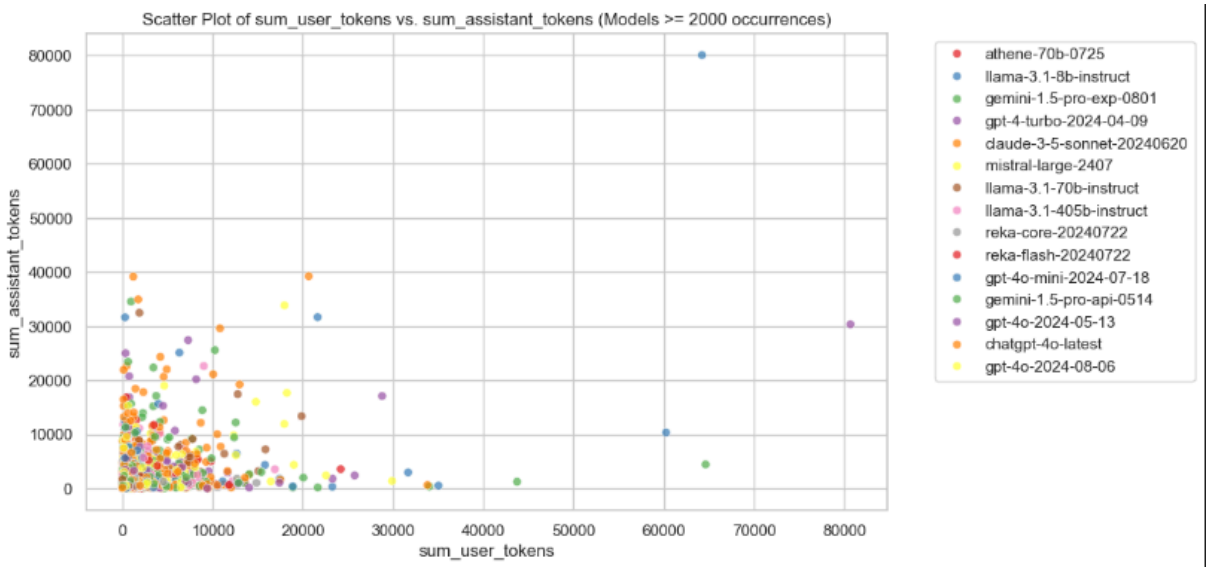
Correlation Matrix of Token Types

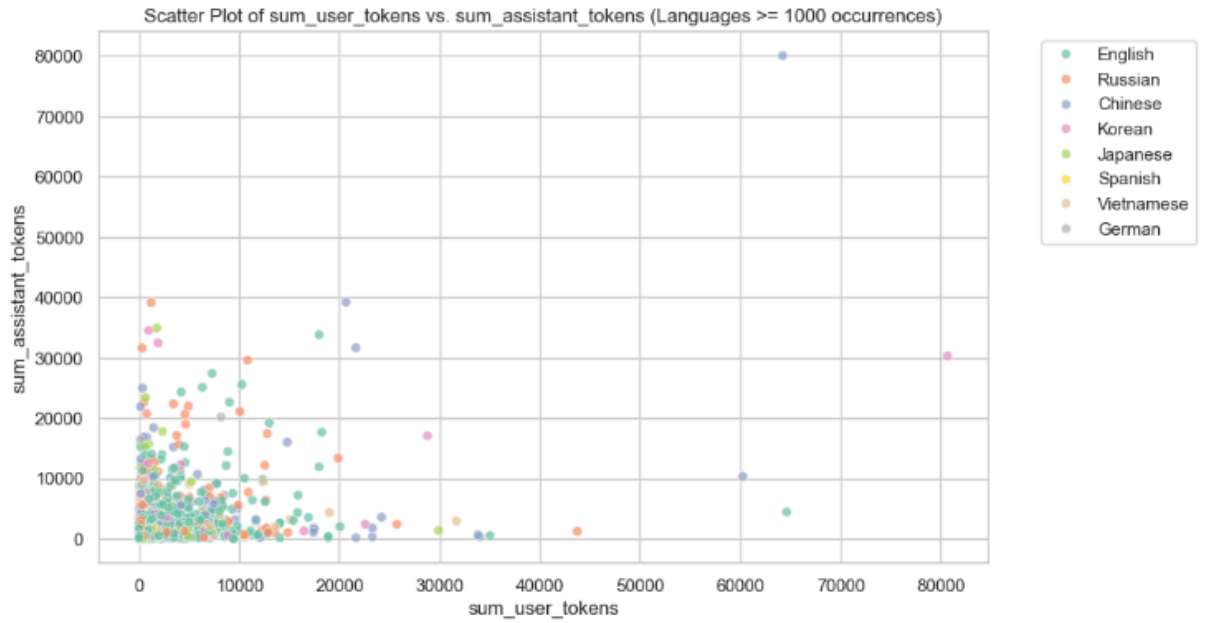


This is a *heatmap* visualizing the *correlation matrix* for the numerical columns: *sum_user_tokens, **sum_assistant_tokens, and **context_tokens. The color intensity and numbers in the cells show the strength of the correlation that goes from -1.0, meaning negative correlation, up to 1.0, meaning positive. It can be seen from this plot that **context_tokens* are highly positively correlated with *sum_user_tokens* with the value of 0.84, while *sum_assistant_tokens* is moderately correlated with the other columns.

This plot is useful for understanding the relationship of token counts in various model development stages and model selection, if the goal is to tune models according to their token efficiency and correlation patterns.

User Tokens vs. Assistant Tokens: Comparison by Model and Language Occurrences

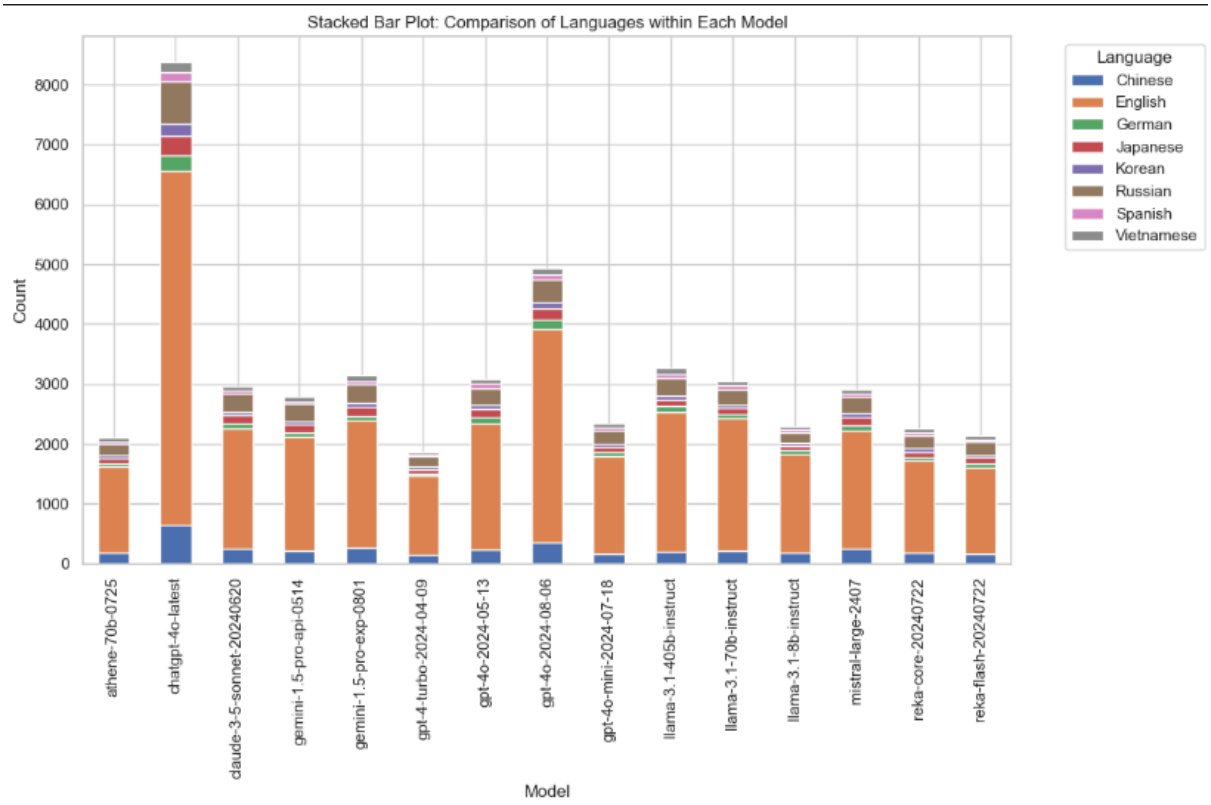




These are scatter plots showing the relationship between user tokens and assistant tokens. The first plot is colored by model, focusing on models with more than 2000 occurrences, while the second plot is colored by language, highlighting languages with more than 1000 occurrences. These plots help identify patterns in token usage across different models and languages.

The plots are useful for model development to optimize token handling based on specific models and languages, and for model selection by determining which models and languages are more efficient or better suited for token-heavy tasks.

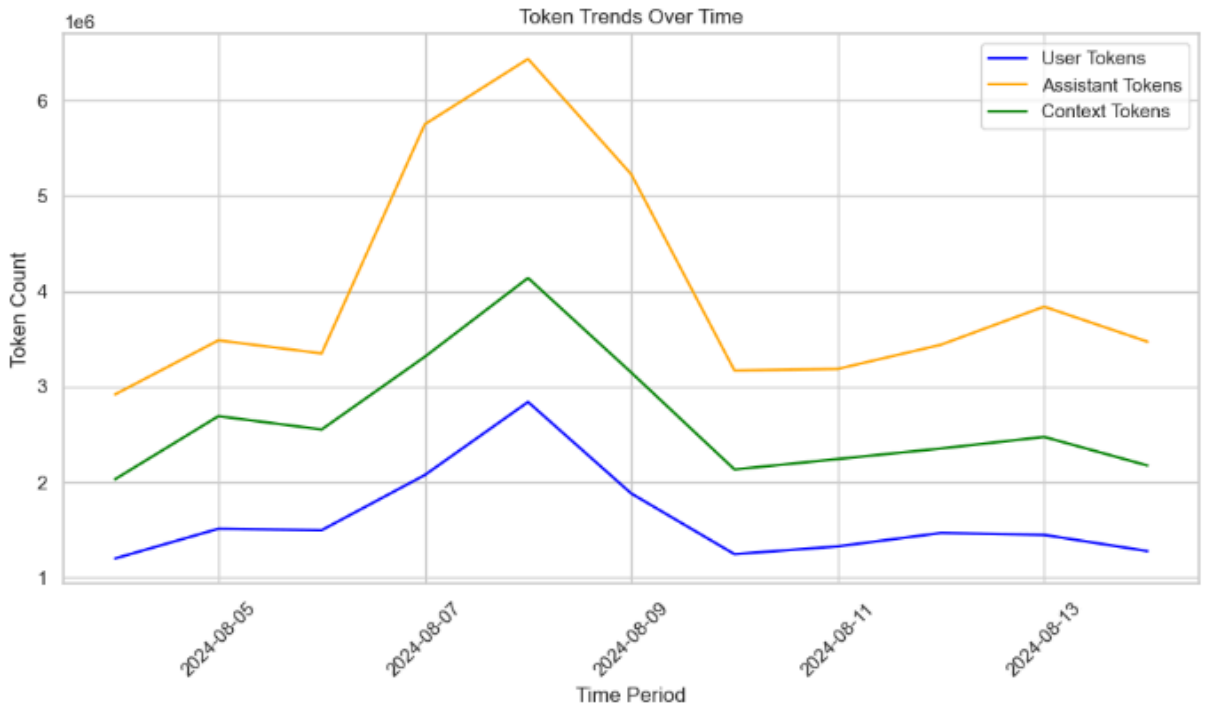
Language Distribution Across Different Models for Multilingual Performance



This is a **stacked bar plot** showing the distribution of languages across different models. Every bar would illustrate the model, while the hue in that bar will show the counts across different languages, for example, English, Chinese, Spanish. The plot shows how frequently each of the languages appears in interactions for each model, with **English** dominating most of the models.

This plot is useful for both model selection by underlining which models perform well with a particular language, and for model development to determine the language handling capability with the aim of ensuring that the models are optimized for a multilingual task.

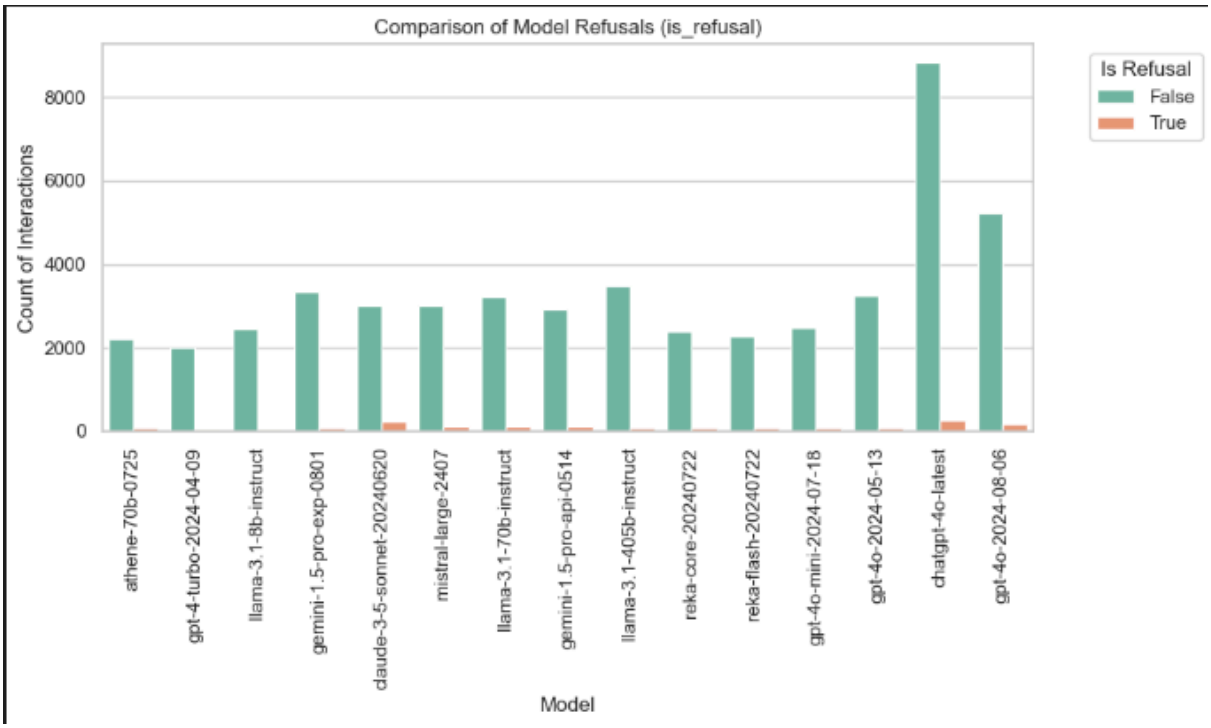
Token Usage Trends Over Time (User, Assistant, and Context Tokens):



This is the *line plot* of the trends of user tokens, assistant tokens, and context tokens over time. The x-axis represents the time periods while the y-axis shows the total token count for each category. The orange line (assistant tokens) peaks the highest indicates that assistant models used more tokens compared to user and context tokens.

This will give insight into token usage patterns, helpful in the development of models by keeping track of token efficiency or model selection in finding out which of those models can handle certain token-intensive tasks efficiently over time. It can also assist in the optimization of token management.

Comparison of Model Refusals (Is_Refusal) for Models with Over 2000 Occurrences



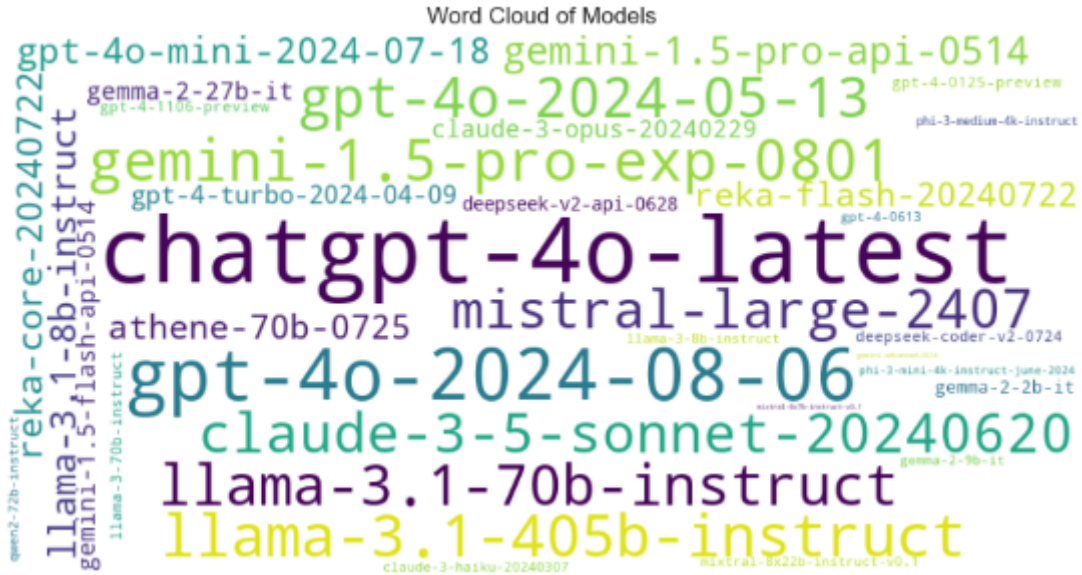
A *count plot* comparing the number of interactions for each of the different models, based on whether the model refused to answer (*is_refusal*). Green bars are for interactions where the model responded, and orange bars represent refusals.

This plot helps to identify models that are most likely to refuse responses. For example, the refusal rate of *chatgpt-4-0-latest* is higher. Hence, this would provide guides on model selection: select models that would be more dependable with respect to response, and in model development: investigate why certain models refuse and how to reduce occurrences of such.

Visualization of Language Distribution

The word cloud above visualizes the most common languages in this dataset. In the word cloud, the size of each name in a language represents the frequency of that name within the data. For example, English, Chinese, and Russian are bigger compared to the other names because they occur more frequently in the dataset. Smaller words like Swedish, Dutch, and Tamil represent less frequency. The following visualization effectively shows the distribution and variation that can occur in the use of language in the dataset

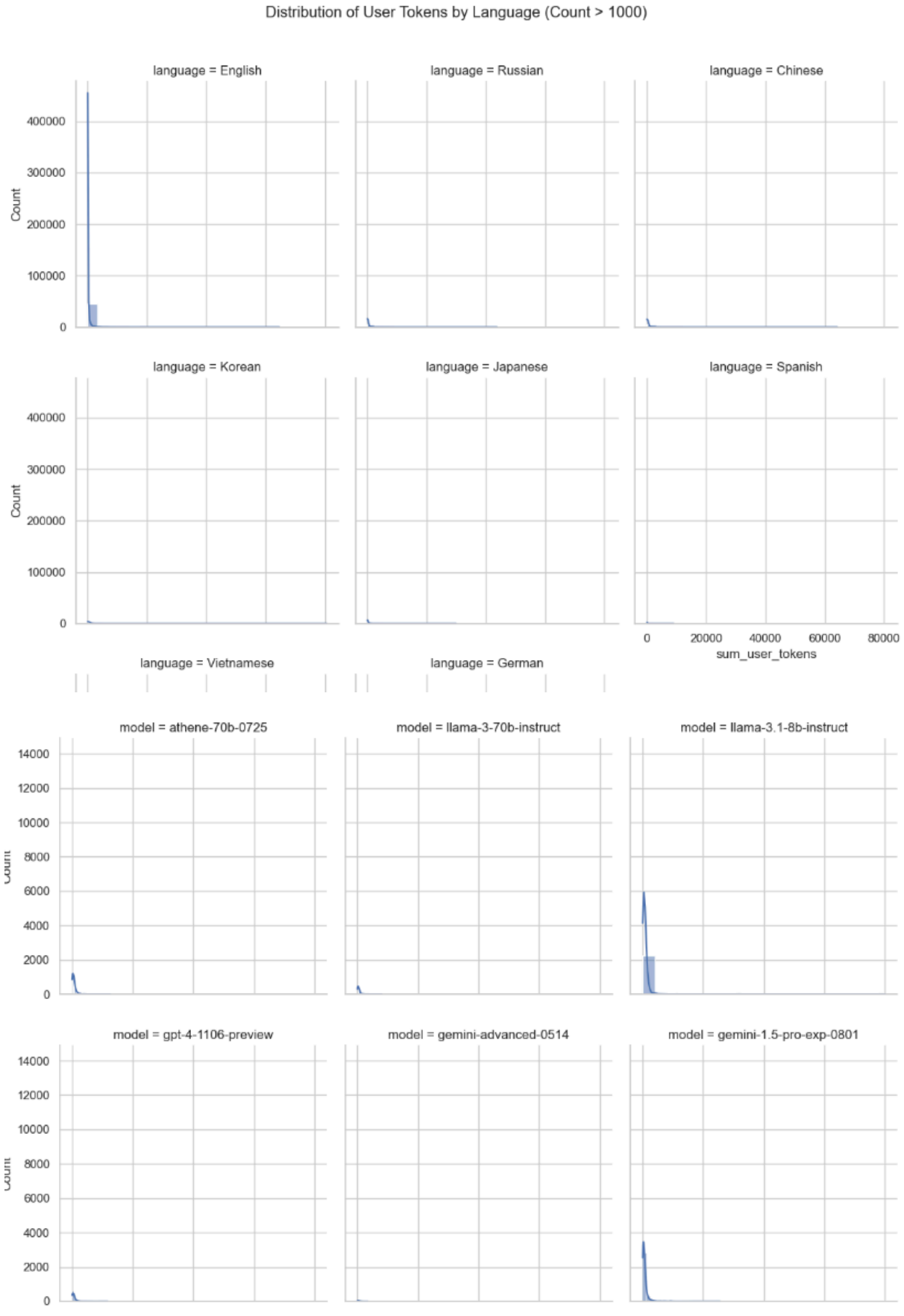
Visualization of Model Distribution



The most frequent, like "chatgpt-4.0-latest", "gpt-4.0-2024-08-06", and "llama-3.1-70b-instruct", are larger since they appear more often.

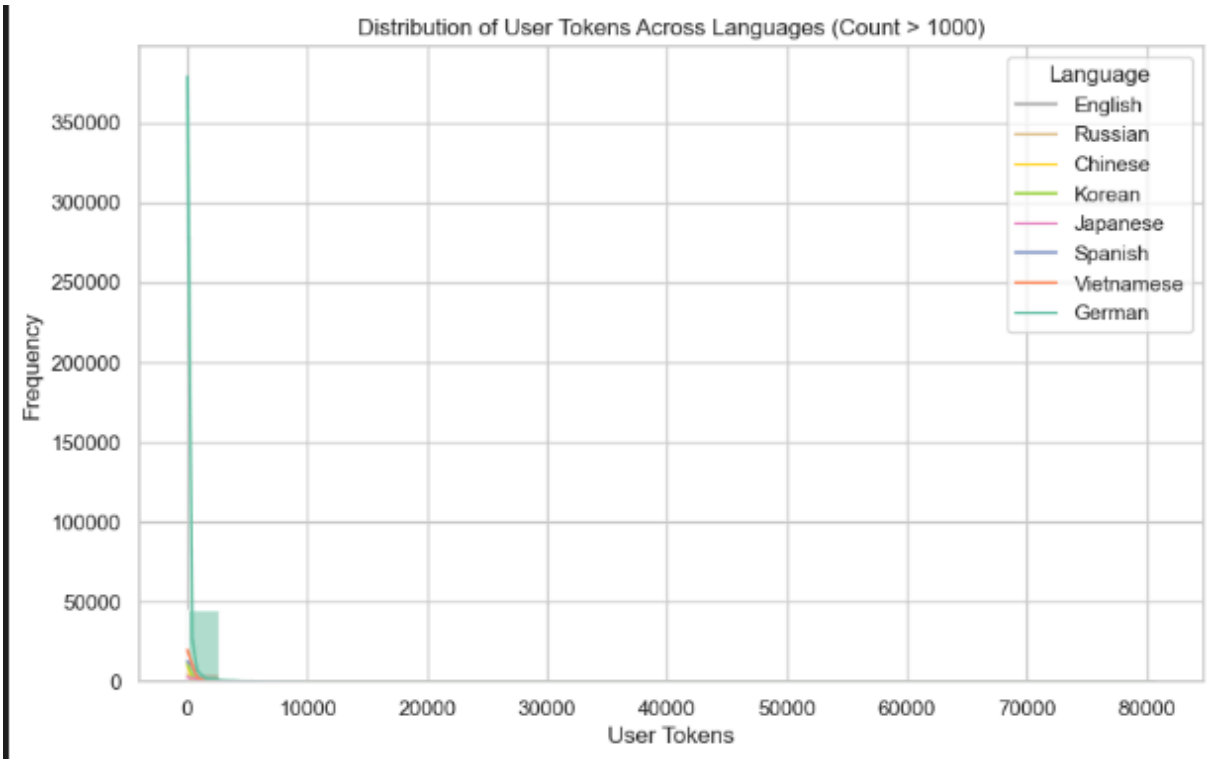
Less frequency words: The less frequent models, such as "gemini-1.5-pro-api-0514" and "deepseek-v2-api-0628", are in smaller text to show how much less they appear compared to others.

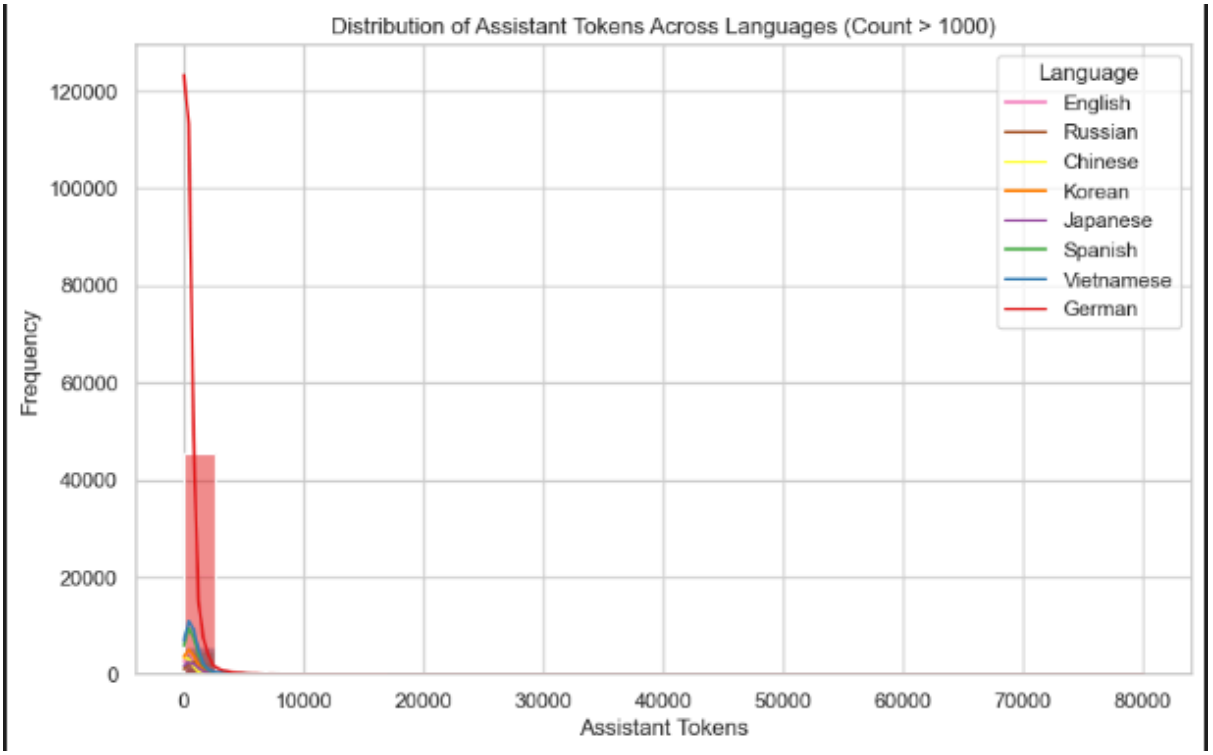
Token Distribution by Language and Model for Languages with Over 1000 Occurrences



Facetgrit provides a multidimensional analysis in the variation between user and assistant tokens to reach deep insights in both language and model behavior.
Distribution of Assistant Tokens by Model(For the languages which have count value of more than 1000) vs Different LLM Models.

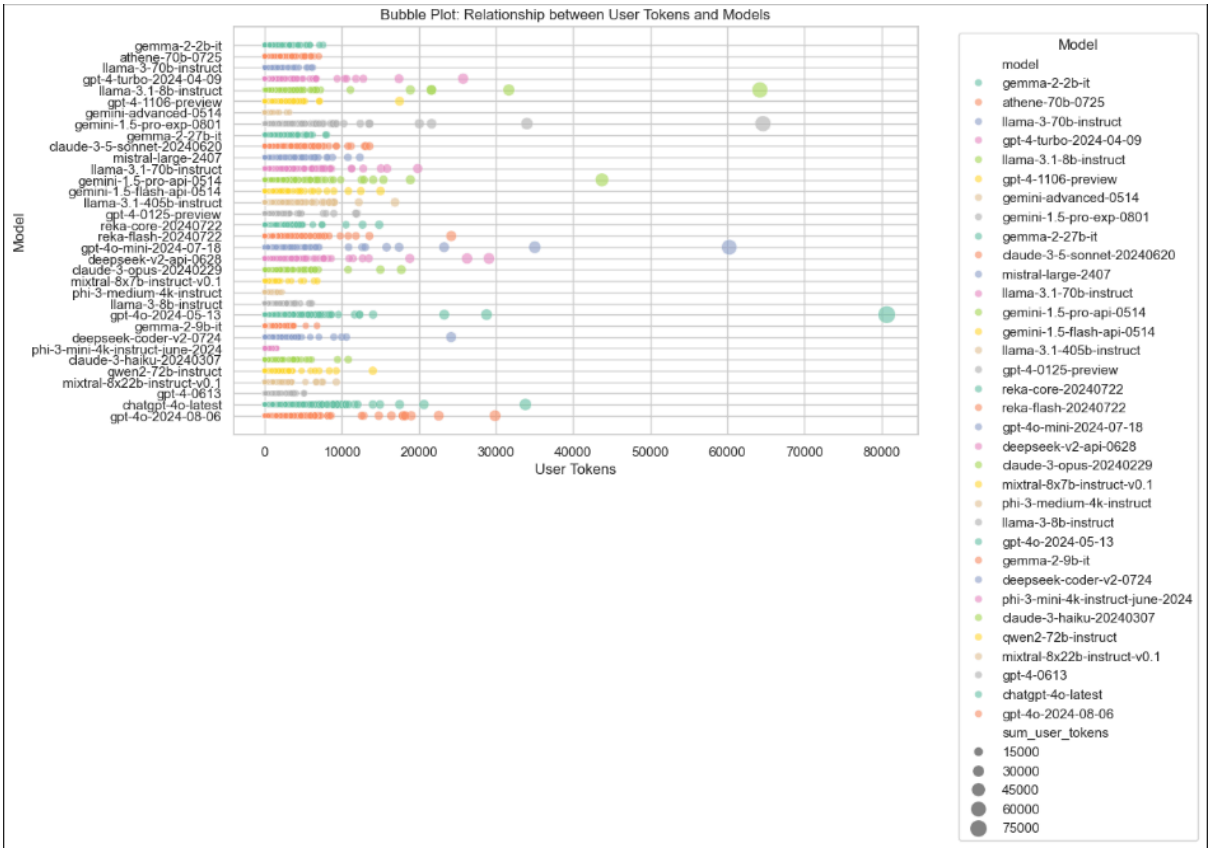
Distribution of User and Assistant Tokens Across Different Languages





These are *histograms* with a kernel density estimate, showing the distribution of user tokens(top plot), and assistant tokens(bottom plot), across languages that have more than 1000 occurrences. Both plots reveal that for the majority of the interactions, regardless of language, just a few tokens are involved, between 0 and 10,000 tokens, with some outliers at higher token counts. Going a step further, we can try to explore language-specific differences or outliers in token usage that might indicate the trends in longer conversations for the next level.

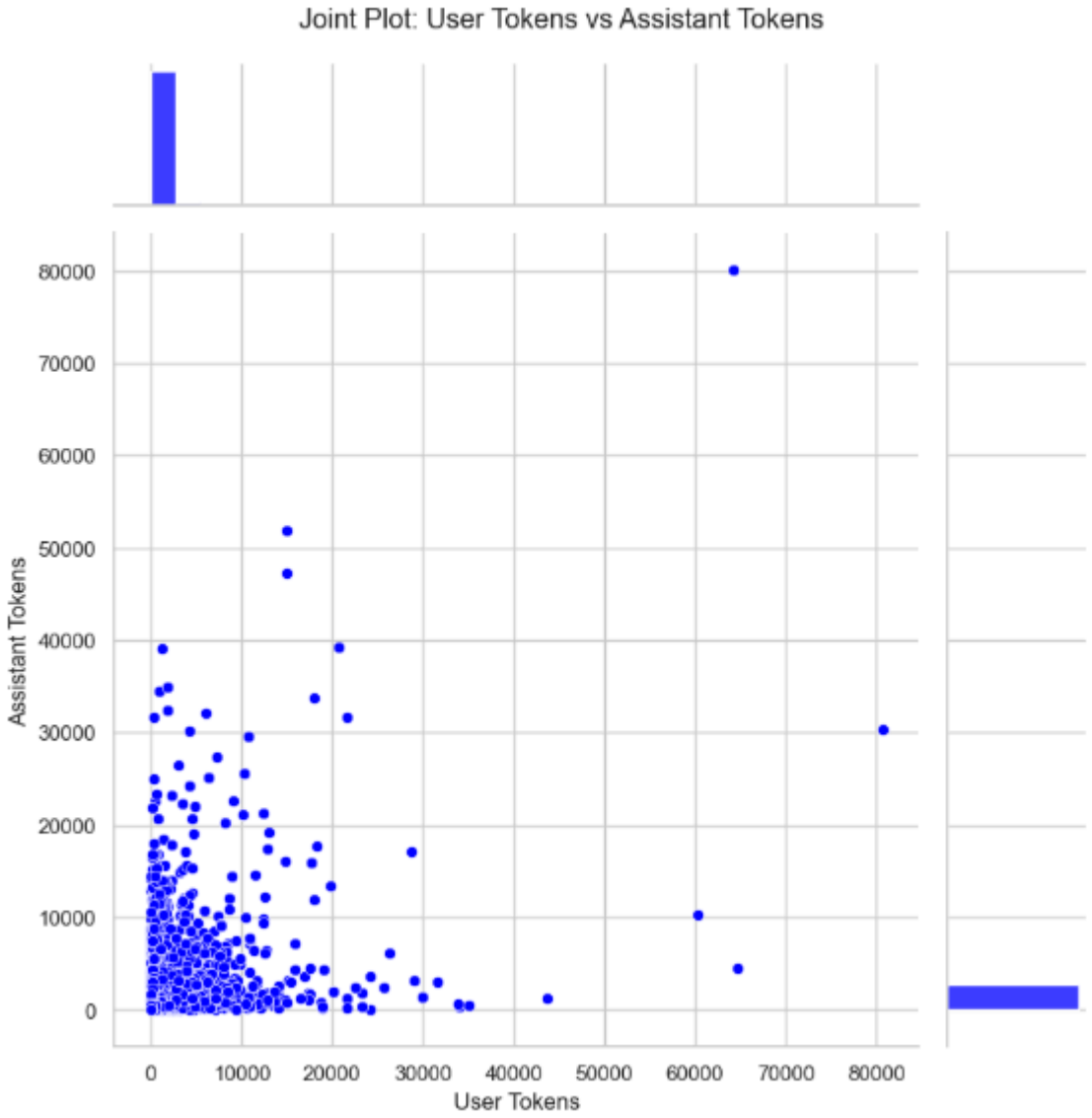
Relationship Between User and Assistant Tokens Across Different Models



The two above *bubble plots* outline the relationship of token usage-(user and assistant) with different models. In each of these plots, the size of each of the bubbles denotes the number of tokens, larger-sized bubbles being indicative of higher token usage. The first plot shows the relationship between user tokens and models, the second between assistant tokens and the same.

These plots are informative for the idea of different model behaviors in light of variable numbers of input tokens and could further drive model development by the identification of models that manage tokens more efficiently, as well as model selection by picking out models suitable for token-intensive tasks.

Comparison of User Tokens vs Assistant Tokens



This is a *joint plot* showing the relation between user tokens and assistant tokens. The scatter plot in the middle shows the distribution of these tokens, and the histograms on top and right show the frequency distributions for user and assistant tokens, respectively.

This plot is useful for model development and model selection, as it shows the relation between how many input-user tokens-are given and how many output-assistant tokens-are returned by a number of models. The plot can help in the optimization of the token usage for efficiency and locating models that handle inputs or outputs with a large count of tokens efficiently.