FINAL REPORT

Name: Shrinidhi Rajesh
ID: 2132508
Class: CSC 583 - Natural Language Processing

Assignment Number: Finals

Collaboration: Utilized StackOverflow, Google, and ChatGPT for assistance
Operating System: Windows 11
Development Environments:
Google Colab: Employed as the primary development environment for code writing, testing, and iterative development.
Visual Studio Code: Utilized for cross-verification and additional testing of code functionality.

General Reflections on Learning:
This project offered an in-depth exploration of Natural Language Processing (NLP) and AI techniques applied to real-world data. The Cleantech Media Dataset, comprising over 20,000 articles, presented a range of challenges and learning opportunities.

Key insights gained from this project include:
Data Management: The project underscored the critical nature of thorough data exploration and preprocessing. Addressing challenges such as duplicate entries, language inconsistencies, and skewed distributions provided valuable experience in data cleaning and preparation techniques.

NLP Fundamentals: The implementation of tokenization and analysis of word frequencies provided practical insights into core text analysis concepts. This hands-on experience enhanced understanding of fundamental NLP processes.

Advanced AI Applications: The development of a Retrieval-Augmented Generation (RAG) system introduced complex AI concepts, demonstrating how retrieval mechanisms can be integrated with generative models to produce more accurate and contextually relevant outputs.

Performance Evaluation: Utilizing various metrics such as RAGAS, Rouge, Perplexity, and BERTScore offered a comprehensive approach to assessing AI model performance, providing a nuanced understanding of evaluation methodologies.

Difficulty of the Assignment:
The difficulty of this assignment was substantial, reflecting its complexity and depth.

The project demanded an extraordinary time commitment, requiring 5 days and 4 sleepless nights to complete. This level of dedication highlights the project's complexity and the steep learning curve involved in mastering advanced NLP techniques. The computational requirements were extensive, consuming approximately 110 computing units. This indicates the resource-intensive nature of processing and analyzing the large Cleantech Media Dataset of over 20,000 articles.

The need to purchase API credits twice underscores both the project's scale and the real-world costs associated with conducting advanced AI research. This aspect adds a practical dimension

to the academic challenge, mirroring resource considerations in professional AI development. The project's breadth, covering data preprocessing, tokenization, implementation of Retrieval-Augmented Generation (RAG), and multiple evaluation metrics, is more akin to a full thesis than a typical course assignment. This scope provided invaluable hands-on experience but also significantly increased the project's complexity. The project involved mastering numerous intricacies, from handling large datasets to implementing sophisticated NLP techniques. This steep learning curve, while challenging, provided a comprehensive understanding of real-world AI application.

An error in handling the evaluation dataset resulted in an additional full day of work and extra computational units. This experience underscored the critical nature of precision in AI projects, where even small mistakes can lead to significant time and resource costs. It demonstrated that AI development, particularly in areas like RAG, requires meticulous attention to detail at every stage.

Despite these challenges, the project proved to be a deeply rewarding experience. The cognitive demands were intense, with work extending into the early hours of the morning and resulting in mental fatigue. However, the sense of accomplishment upon completion was significant.

The project offered a unique opportunity to apply theoretical knowledge to practical problems, providing insights into the complexities of working with real-world data and advanced AI techniques. The experience gained from this project goes beyond academic learning, offering a glimpse into the demands and rewards of professional AI development.

Introduction

The Cleantech Media Dataset, developed by Anacode, is a comprehensive repository of web articles centered on clean technology innovations. Spanning January 2022 to October 2024, it comprises 20,122 articles sourced from diverse platforms such as energy-xprt and pv-magazine. This dataset offers a wealth of information on sustainable technologies, making it a valuable resource for research and analysis in the cleantech domain.

Dataset Characteristics

Designed to address the challenges of analyzing technological innovations in the cleantech sector, the dataset is meticulously structured. Each article is presented in the content column as a collection of paragraphs, supplemented with detailed metadata, including publication dates, source information, and URLs. This organization facilitates advanced analyses of market trends and technological developments, supporting actionable insights for the cleantech industry.

Research Context

The dataset gained prominence at the Next-Gen Cleantech Solutions Workshop during SwissText 2024. The workshop emphasized the pressing need for innovative methods to process the vast amounts of patent and media data in the cleantech sector, which are difficult to analyze manually. Natural Language Processing (NLP) and Large Language Models (LLMs) were recognized as key tools for accelerating cleantech innovation, enabling stakeholders to extract trends and understand market requirements with greater efficiency and precision.

Project Objectives

This research project leverages the Cleantech Media Dataset to develop a Retrieval-Augmented Generation (RAG) system tailored to the clean technology domain. The primary objectives of the project include:

- Implementing sophisticated NLP techniques for effective text processing.

- Implementing and comparing multiple text splitting techniques, including NLTKTextSplitter, to optimize document chunking.
- Experimenting with various embedding models ("mini", "bge-m3", "gte") to enhance information retrieval from cleantech articles.
- Developing and testing different RAG prompt templates to improve the quality of retrieved information.
- Evaluating the impact of varying chunk retrieval numbers on system performance.
- Applying multiple evaluation metrics such as RAGAS, Rouge, Perplexity, BERTScore to assess the system's performance comprehensively.
- Conducting qualitative evaluation to provide deeper insights into the system's effectiveness in summarizing cleantech information.

Evaluation Framework

The project incorporates a complementary evaluation dataset, consisting of 23 high-quality instances, to rigorously assess the RAG system's retrieval capabilities. This evaluation set, structured with columns for question, relevant_text, and answer, enables a systematic performance assessment through both quantitative metrics and qualitative analysis.

What to Expect

By utilizing the Cleantech Media Dataset and implementing advanced computational methods, this project aims to develop a sophisticated Retrieval-Augmented Generation (RAG) system that effectively addresses the challenges of analyzing vast amounts of cleantech information. The project focuses on extracting actionable insights from the dataset, which consists of 20,122 articles covering various aspects of clean technology.

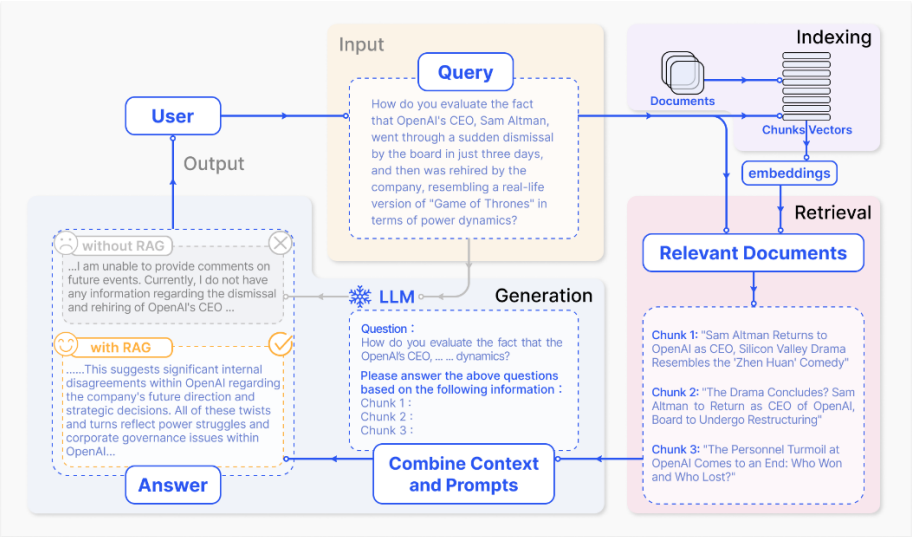
The outcomes of this project are intended to significantly enhance understanding within the cleantech sector by providing researchers and industry professionals with efficient tools for data analysis. By leveraging Natural Language Processing (NLP) techniques and exploring different embedding models and retrieval strategies, the project seeks to facilitate informed decision-making and promote sustainable technological innovation. Ultimately, this work aspires to bridge the gap between extensive cleantech resources and practical applications that can drive advancements in environmental solutions.

Retrieval-Augmented Generation (RAG) in NLP and Its Application to the Cleantech Project

Retrieval-Augmented Generation (RAG) is an advanced Natural Language Processing technique that combines information retrieval with text generation. It enhances the capabilities of large language models by allowing them to access and utilize external knowledge sources. In NLP, RAG offers several benefits: it improves the accuracy and relevance of generated responses, enables models to work with up-to-date information, and helps in handling domain-specific queries more effectively.

For this cleantech project, RAG is particularly valuable due to the nature of the dataset and the project goals. The Cleantech Media Dataset contains 20,122 recent web articles on clean technologies, spanning from January 2022 to October 2024. RAG allows the system to efficiently navigate this large corpus, retrieving relevant information based on user queries about clean technology. By implementing RAG, the project can generate more accurate and context-specific summaries of cleantech innovations, market trends, and technological developments. This approach is crucial for processing the vast amount of patent and media data in the cleantech sector, which is challenging to analyze manually. RAG enables the project to capture development trends effectively,

provide insights into market requirements, and ultimately contribute to accelerating innovation in sustainable technologies



Plot 1: RAG Framework

Project Setup and Data Loading

- Imported necessary libraries: Loaded required libraries for data processing and NLP tasks.
- Installed dependencies: Installed essential packages such as torch, pandas, and others for project requirements.
- Mounted the drive: Mounted the Google Drive to access datasets stored on it.
- Loaded train and evaluation datasets: Loaded the training and evaluation datasets from CSV files stored on the drive for analysis.

Training Dataset Columns	Evaluation Dataset Columns
Column Name	
title	question_id
date	question
author	relevant_text
content	answer
domain	article_url
url	

Table 1: Column Names

Exploratory Data Analysis

Training Dataset:

The initial analysis of the training dataset revealed a total of 20,111 records distributed across 6 columns (as shown in Table 1). Among these, the author column was found to contain no values and was subsequently removed from the dataset.

Analyzing the Article Published Dates

Further examination of the publication dates uncovered a notable anomaly in the daily article counts between May 4, 2023, and June 15, 2024. The following outlier dates and their corresponding article counts were identified:

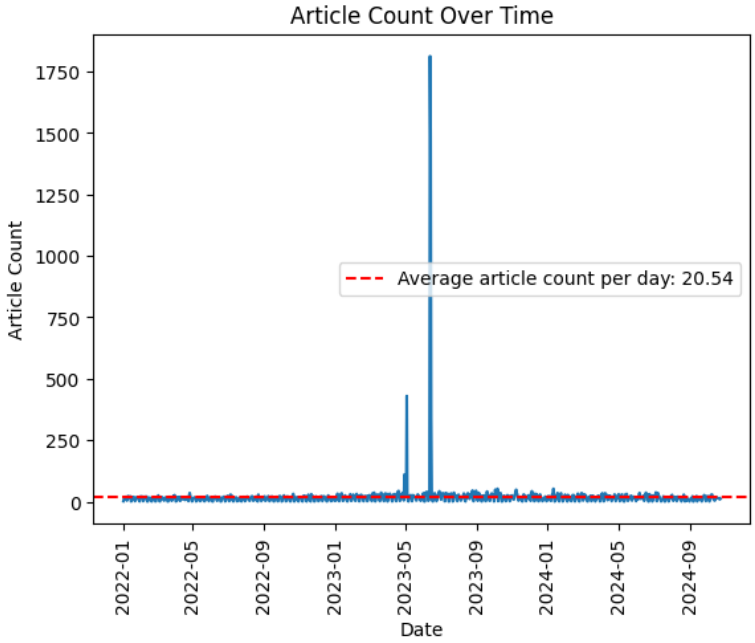
June 13, 2023: 1812 articles

June 14, 2023: 1493 articles

May 4, 2023: 430 articles
June 15, 2023: 340 articles

This irregularity disrupts the otherwise consistent daily article counts. Although the exact cause remains undetermined, one hypothesis is that it might be related to the dataset's scraping process, where missing dates were assigned a default value. Another hypothesis might be that the dataset might have been scraped in batches, causing articles published over multiple days to be recorded under a single date.

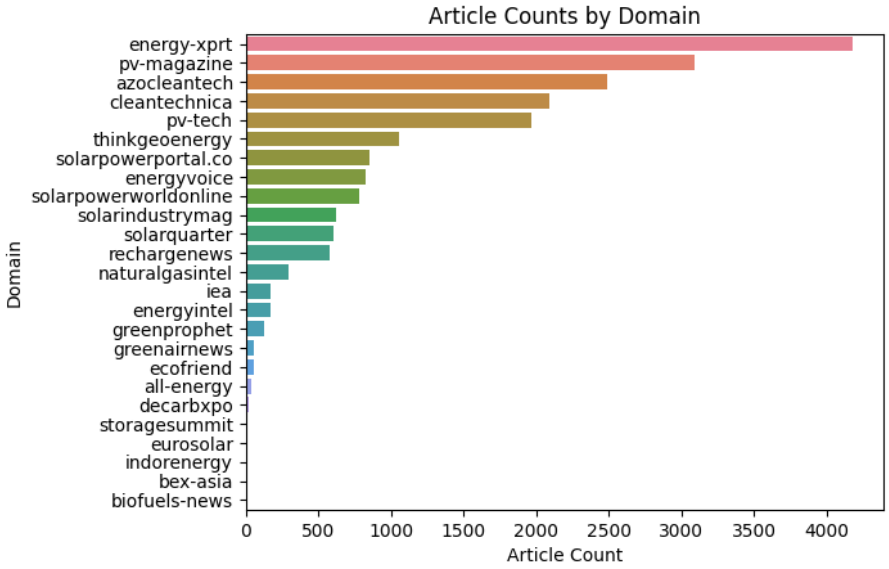
Given that the publication date column is not critical for the Retrieval-Augmented Generation (RAG) pipeline, it was excluded from further analysis to streamline the dataset.



Plot 3: Date Distribution of Training Set

Analyzing Domain Counts

To analyze the distribution of articles across various sources, I performed a count of articles by their respective domains. The result is displayed in Plot 4.



Plot 4: Domain Distribution of Training Set

The domain distribution reveals a clear skewness, with the top three domains—energy-xprt, pv-magazine, and azocleantech—accounting for a disproportionate number of articles (over 10,000 articles combined). In contrast, the other domains show much lower counts, indicating that the dataset is heavily dominated by a few major sources. This skewness suggests that the content from these three domains might be overrepresented, which could impact the analysis or model training by introducing bias. Therefore, it may be important to consider this skewness in future steps, possibly by adjusting for domain balance if necessary.

Counting Duplicated Titles

The dataset contains 95 duplicated article titles, indicating 95 instances where multiple articles share the same title. This suggests potential redundancy in the dataset that could be addressed during data cleaning or modeling.

Extracting and Sorting Duplicated Titles

Duplicated titles were extracted and sorted to understand the pattern of repetition. The analysis revealed that the duplicated content for these titles is minimal, with only 6 instances of duplicated content. This points to a small portion of the dataset that could be cleaned to remove redundancy.

Article Content Duplication

The article content was joined into a single string for each article, resulting in 43 duplicated articles based on content. This further emphasizes the need to handle repeated entries to maintain the quality and uniqueness of the dataset.

Language Detection

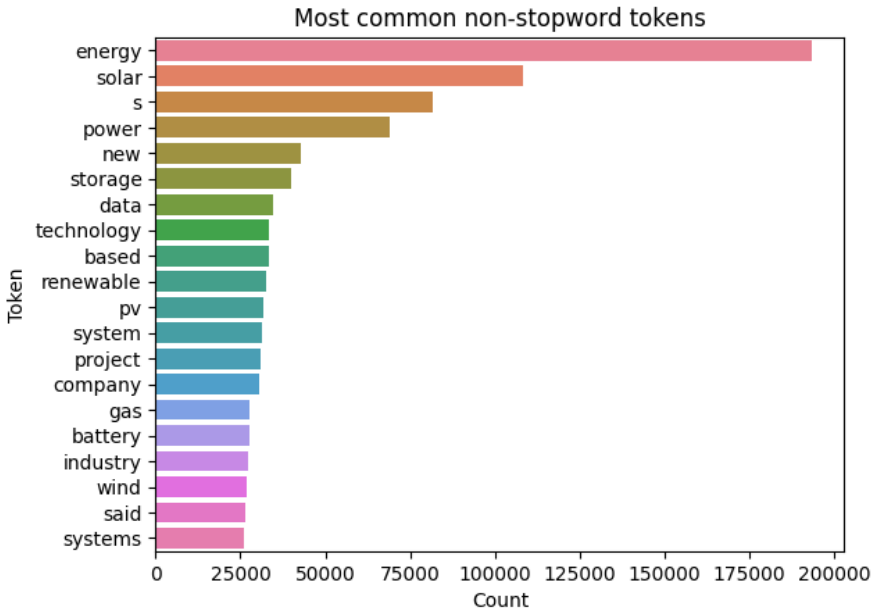
Language detection was performed on the article content, and the detected language was added as a new column, lang. The results showed that the majority of articles are in English (en), with only 3 articles in German (de), 1 in Russian (ru). Given the predominance of English articles in the dataset, the dataset was filtered to retain only the English-language articles, reducing the dataset size to 20,107 articles. This ensures that the dataset remains consistent with the primary language of the research and prevents the inclusion of articles in other languages that might not align with the analysis objectives.

Note on Language Filtering

Filtering for English articles is essential for the success of Natural Language Processing (NLP) tasks, such as prompt engineering and Retrieval-Augmented Generation (RAG), as many models are optimized for English. Non-English articles pose several challenges, including the need for additional language-specific processing and the risk of reduced model performance when these articles are used. Since the majority of available resources, embeddings, and pre-trained models are fine-tuned for English, it is crucial to ensure the dataset is aligned with the language capabilities of the models. Removing non-English articles helps maintain consistency, improves the quality of analysis, and ensures better results when applying advanced NLP techniques.

Tokenization and Analysis of Article Content

Tokenization was performed on the article content using the spaCy English model, converting each article's raw text into individual tokens. Non-alphabetic tokens, such as punctuation, were filtered out to focus on meaningful words, and all tokens were then converted to lowercase for uniformity. This step ensured that words like "Energy" and "energy" were treated as the same. The occurrence of each alphabetic token was counted to identify the most common words, revealing frequent terms related to energy, climate, and technology. Stopwords, such as "the" and "and," were removed to focus the analysis on more significant terms. A bar plot was then generated, plot 5, to visualize the most common non-stopword tokens, further confirming the dataset's relevance to cleantech topics.



Plot 5: Most common word counts

The prominence of words such as "energy," "solar," and "power" at the top of the token frequency list indicates that the training set is highly relevant to the cleantech sector. This reinforces the suitability of the dataset for a project focused on clean technology, as these key terms align with the domain's core themes.

Note on Tokenization and Its Importance

Tokenization plays a pivotal role in NLP tasks by converting raw text into manageable pieces that can be easily analyzed. It is the first step in most NLP pipelines because it breaks down unstructured text

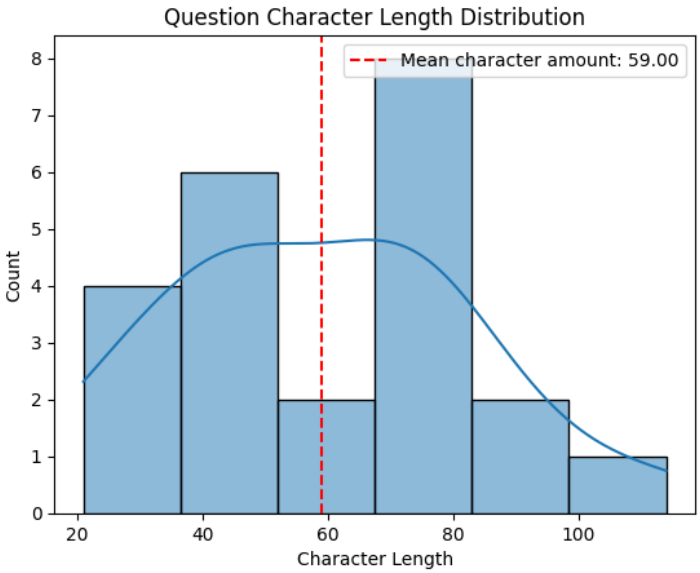
into individual elements (tokens) that can be processed further. In this project, tokenization helps identify the frequency and importance of words in cleantech articles, which is crucial for building models that understand the content of the articles. Effective tokenization allows for better handling of text data, ensuring that relevant words are highlighted while irrelevant ones (such as stopwords or punctuation) are excluded. The importance of tokenization also extends to the development of Retrieval-Augmented Generation (RAG) model. By converting articles into a structured form of tokens, it becomes easier for the system to retrieve meaningful information, match user queries with relevant data, and generate more accurate summaries or answers based on the cleantech domain. Thus, proper tokenization helps improve the efficiency and effectiveness of downstream tasks, contributing to the overall success of the project.

Structuring

The creation of a structured folder system is an essential part of managing the project's data and ensuring smooth workflow. In this project, the path to the main data folder in my Drive is defined first, followed by the creation of subfolders, labeled "silver" and "gold" which are used to store intermediate data outputs such as the datasets, embeddings, LLM results. This step is crucial for maintaining an organized directory structure, ensuring that files are properly stored and easily accessible throughout the project's different stages. A well-structured folder system such as this helps in efficient data management and facilitates seamless collaboration and scaling as the project progresses.

Evaluation Dataset:

The evaluation dataset, human_eval_df, contains a total of 23 entries, with five columns as mentioned in Table 1. After performing some preprocessing, such as renaming columns for clarity and dropping unnecessary ones, the dataset was cleaned and organized. A histogram was created to visualize the distribution of character lengths in the "question" column, revealing the average length of questions, plot 6. The character length seems to be between 20 and 120, with mean amount of approximately 60.



Plot 6: Question length distribution of the evaluation dataset

URLs in the evaluation dataset were normalized to ensure consistency with the articles dataset, and any mismatched URLs were updated. This resulted in a refined evaluation set that aligns with the articles dataset, ensuring that almost all the URLs in human_eval_df now match those in articles_df.

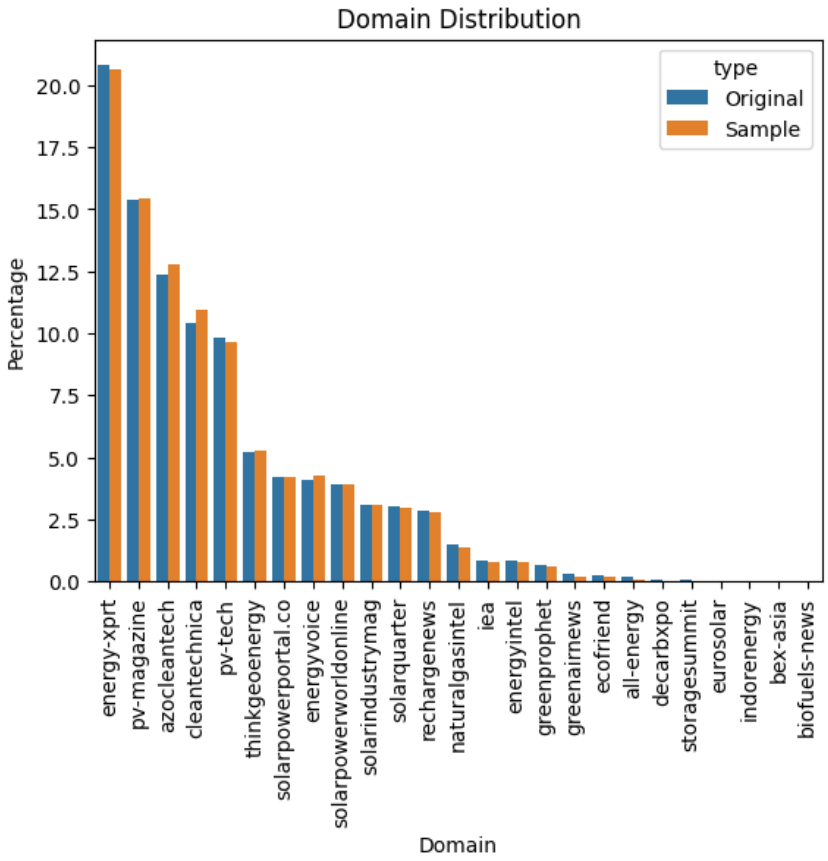
Subsampling

To optimize processing time and manage computational costs, the dataset was subsampled to 1000 articles. This subsampling ensures a balance between runtime efficiency and meaningful analysis. Given the skewed distribution of articles across publishers, stratified sampling was used to maintain representativeness. Stratification ensures that the sample retains the proportional representation of domains as in the original dataset. Additionally, articles linked to the evaluation data were identified and retained in the subsample to maintain consistency for evaluation purposes.

Stratified sampling was performed on the domain column using a custom function, ensuring proportional representation in the selected sample. The sample was further refined by excluding evaluation articles already present in the subset. These excluded articles were then concatenated with the sampled dataset to create a final dataset of 1004 articles. This final dataset comprises a combination of representative samples and evaluation-specific articles, ensuring alignment with project goals while preserving data diversity.

Post-sampling, the content column was processed for uniformity. Line breaks were replaced with spaces, and any string representations of lists were converted to proper lists. The content items were then formatted for clarity, with each item displayed on a new line. This preprocessing step improved the readability and usability of the content data for subsequent analysis. The updated DataFrame was saved as a CSV file, ensuring data preservation for future use.

Finally, the distribution of domains in the original and subsampled datasets was visualized as in plot 7. By calculating the relative percentages of each domain and plotting them side by side, it was confirmed that the stratified sampling process effectively preserved the original distributional characteristics. This ensures that the subsample is representative of the broader dataset while reducing computational costs.



Plot 7: Domain distribution comparison between original and the sub sampled data

What is Chunking?

Chunking is the process of dividing large texts into smaller, meaningful segments or chunks. This approach helps in retaining coherence while making the text manageable for various tasks like processing, summarization, or retrieval. Chunking is especially useful for dealing with large or unstructured documents that are difficult to process as a single unit.

Purpose of Chunking in Retrieval-Augmented Generation (RAG)

In RAG, chunking plays a vital role in enabling efficient information retrieval. Instead of analyzing an entire document, smaller chunks are indexed and used to generate contextually accurate responses. This approach improves retrieval accuracy, reduces computational effort, and maintains contextual relevance across segments, ensuring that the outputs generated by models are precise and context-aware.

Chunking Strategies Used

Recursive Chunking

Recursive chunking involves splitting text hierarchically using predefined separators such as paragraphs, sentences, or words. The process continues until each chunk meets a specific size limit. Overlapping chunks are also created to maintain continuity across segments. This method is particularly useful for structured documents and ensures chunks are logically coherent and manageable in size.

NLTK Chunking
NLTK chunking uses syntactic rules to segment text, often identifying linguistic units like phrases or sentences. It employs grammar-based splitting, making it ideal for tasks that require syntactic coherence, such as sentence-level retrieval or analysis. By leveraging natural language processing principles, this method produces grammatically meaningful chunks that align with linguistic structures.

Semantic Chunking
Semantic chunking splits text into segments based on the semantic boundaries between sentences or phrases. This method uses embedding models to calculate sentence similarity, identifying points where semantic differences occur. By creating chunks that are semantically coherent, this strategy improves the relevance of retrieved information, making it highly effective for unstructured or conceptually complex documents.

As the project code develops, key questions are addressed to refine and optimize the chunking process, ultimately improving the effectiveness of Retrieval-Augmented Generation (RAG). Two main points were implemented and tested in parallel to enhance the solution:

Adding In-Document Chunk IDs:
To enhance metadata for each chunk, an in-document chunk ID was introduced, providing a 0-based sequential identifier for chunks within each document. This allows for clear tracking and identification of chunks relative to their source document.

Using NLTKTextSplitter:
In addition to the existing splitters, the NLTKTextSplitter was implemented using a customized get_nltk_splitter function. This linguistically aware splitter respects sentence or paragraph integrity during segmentation, making it especially suited for well-formatted, natural language text. Two variations of the splitter were created with chunk sizes and overlaps a(256, 64) and (1024, 128).

Understanding Embeddings in NLP
Embeddings are dense vector representations of text, where words, phrases, or entire sentences are mapped to points in a high-dimensional space. These representations capture semantic relationships, enabling models to compare and process language meaningfully. In natural language processing (NLP), embeddings are foundational for tasks like information retrieval, semantic search, machine translation, and text classification. Hugging Face provides access to pre-trained embedding models, making it easier to incorporate state-of-the-art language representations into projects.

The Three Embedding Models Utilized
Mini (MPNet-Based Model)
The Mini model leverages the all-mpnet-base-v2 architecture, which produces embeddings of 768 dimensions. MPNet uses a self-supervised contrastive learning objective trained on over a billion sentence pairs. It excels at understanding semantic relationships in text, making it well-suited for tasks like semantic similarity, clustering, and information retrieval. Its combination of efficiency and accuracy makes it a popular choice for general-purpose NLP tasks.

BGE-M3 (BAAI/BGE-M3)
The BGE-M3 model, developed by BAAI, focuses on generalized sentence embeddings with dense vector outputs of 1024 dimensions. Designed for multi-language contexts, it facilitates efficient

retrieval and deep semantic understanding. This model is particularly useful in multilingual applications where precise sentence-level understanding is crucial, such as cross-lingual search or document similarity detection.

GTE (Alibaba GTE Model)
The GTE (General Text Embedding) model, gte-base-en-v1.5 by Alibaba, produces embeddings of 768 dimensions. It is a multilingual transformer model designed for tasks like semantic textual similarity and information retrieval. Its dense embeddings are optimized for versatility, making it an excellent choice for projects involving diverse data and varied NLP tasks, including classification and semantic search.

Why These Models Were Chosen
Addressing question d). The selection of these three models reflects the author's intention to balance performance, dimensionality, and domain versatility. The Mini (MPNet) model was likely included for its strong accuracy in sentence-level semantic understanding and its computational efficiency. The BGE-M3 model provides a higher-dimensional alternative that excels in multilingual and cross-lingual tasks, making it a valuable asset for multi-language datasets. Finally, the GTE model combines multilingual support with general-purpose capabilities, ensuring robust performance across a wide range of NLP tasks.

Purpose and Goals
By using these three distinct models, the author ensures comprehensive coverage of various NLP requirements, such as semantic similarity, efficient retrieval, and multilingual analysis. The diverse embedding sizes (768 and 1024 dimensions) offer flexibility in model selection based on specific computational or application needs. This thoughtful combination aligns with goals like improving retrieval performance, enhancing semantic understanding, and ensuring adaptability to different domains and languages.

Chunking Strategies and Implementation
Chunking as mentioned before is an essential preprocessing step in natural language processing (NLP), where large text documents are divided into smaller, manageable segments.

For Recursive Chunking, two recursive splitters were defined, characterized by fixed chunk sizes and overlaps:
Recursive 256 Splitter: Configured with a chunk size of 256 and an overlap of 64.
Recursive 1024 Splitter: Configured with a chunk size of 1024 and an overlap of 128.
Recursive chunking ensures that adjacent chunks share overlapping segments of text, maintaining continuity and context for downstream tasks.

A semantic chunking strategy was implemented using dense embeddings generated by the "mini" model (MPNet-based). In this approach, sentences are grouped into chunks based on semantic similarity, determined by the cosine distance between consecutive sentences. The threshold for grouping is defined using a percentile-based measure, ensuring that chunks represent cohesive semantic units.
Two NLTK-based splitters were defined, using linguistic boundaries to create chunks:
NLTK 256 Splitter: Configured with a chunk size of 256 and an overlap of 64.
NLTK 1024 Splitter: Configured with a chunk size of 1024 and an overlap of 128.

NLTK-based splitting preserves sentence boundaries, ensuring that chunks consist of complete sentences.

Chunking Process and Metadata
Chunks were created by iterating over rows in a dataset and applying the defined splitters. For each document, text from the title and article content was combined and split into chunks. Metadata fields were added to each chunk, including:

title: Title of the document.
url: Source URL of the document.
domain: Domain of the source.
id: A unique identifier for each chunk across the dataset.
in_document_chunk_id: A 0-based identifier for the chunk within its document.

Table 2: Metadata

This metadata as mentioned in table 2 ensures traceability and facilitates downstream processing.

Chunking Results
The number of chunks created for each splitter demonstrates the differences in chunking strategies:

Chunking Strategy	**Number of Chunks**
Recursive 256 Splitter	26,630
Recursive 1024 Splitter	6,018
Semantic Splitter	3,293
NLTK 256 Splitter	25,127
NLTK 1024 Splitter	6,059

Table 3: Chunking Results

These results highlight how different splitting strategies impact the number and size of generated chunks, reflecting variations in their design and intended use cases.

Analyzing the Chunks:
The analysis involved splitting a given text into chunks using different chunking methods to assess how they affect content processing. Several splitting strategies were applied, each with distinct characteristics:

First Chunk – With No Splitter
The first chunk in the original dataset consists of the full unmodified content, without any chunking applied. This chunk represents the entire article text as a single, continuous block, including all paragraphs and information. No segmentation has been performed at this stage, which results in a long chunk of text that may be unwieldy for analysis or processing. This chunk does not account for text structure or logical boundaries, making it harder to extract meaningful segments for analysis.

Recursive 256 Splitter (Chunk Size: 256, Overlap: 64)
This chunking strategy divides the text into segments of 256 characters, with a 64-character overlap between consecutive chunks. The key purpose of this overlap is to ensure that some context is preserved between chunks, which is especially important for tasks where understanding the flow of information between chunks is critical. In this case, each chunk is relatively short, making the content more manageable for analysis. However, it may lack deeper context since it cuts off at relatively small intervals of the text.
Example Output:
The chunk contains only part of the article (e.g., "Charging Ahead: The UK's Electric Vehicle Revolution"), with the content cut off at 256 characters.

Recursive 1024 Splitter (Chunk Size: 1024, Overlap: 128)
This method uses a larger chunk size of 1024 characters, with a 128-character overlap between consecutive chunks. The larger size allows for a richer portion of the text to be included in each chunk, providing more context and making it easier to capture key points or larger sections of content. The overlap continues to ensure that there is no loss of continuity between chunks, allowing for smoother transitions when analyzing or processing the text in parts.
Example Output:
The chunk in this case captures a broader section of the article (e.g., "Charging Ahead: The UK's Electric Vehicle Revolution..."). The larger chunk size allows for more context and understanding, but it still contains an overlap to maintain flow.

Semantic Splitter (Based on Sentence Boundaries)
Unlike the fixed-size splitters, the semantic splitter divides the text based on natural sentence boundaries. This strategy ensures that each chunk corresponds to a meaningful segment of text, such as a paragraph or group of related sentences. By focusing on the semantic structure of the text, this method produces chunks that are contextually coherent and logical. This is particularly useful for tasks such as document classification, summarization, or question answering, where preserving meaning is critical.
Example Output:
The chunk might include entire sentences or a few paragraphs that form a coherent unit, like: "Change is sweeping the highways of the United Kingdom. Being responsible for 88% of passenger miles and 79% of freight traffic, England's highways are crucial."

NLTK 256 Splitter (Chunk Size: 256, Overlap: 64)
This chunking method, implemented with NLTK (Natural Language Toolkit), operates similarly to the recursive 256 splitter, dividing the text into chunks of 256 characters with a 64-character overlap. The NLTK version may slightly differ in how it tokenizes or processes the text compared to other methods. While the chunks remain small, this strategy ensures that overlapping segments are included to maintain context between consecutive chunks. Like the recursive splitter, this approach prioritizes smaller, more manageable chunks of text.
Example Output:
The chunk contains part of the content with a 256-character limit (e.g., "Charging Ahead: The UK's Electric Vehicle Revolution..."), but might differ slightly due to how NLTK handles tokenization and chunk boundaries.

NLTK 1024 Splitter (Chunk Size: 1024, Overlap: 128)
The NLTK 1024 splitter is similar to the recursive 1024 splitter, but it uses NLTK's text processing tools. The chunk size is set to 1024 characters, with a 128-character overlap. This method captures larger portions of the text, ensuring that each chunk contains a more substantial context. The overlap between chunks helps maintain continuity across the text, which is especially important for analyzing longer pieces of content.
Example Output:
The chunk includes a larger portion of the article (e.g., "Charging Ahead: The UK's Electric Vehicle Revolution..."). This allows for more comprehensive analysis while maintaining overlap for contextual integrity.

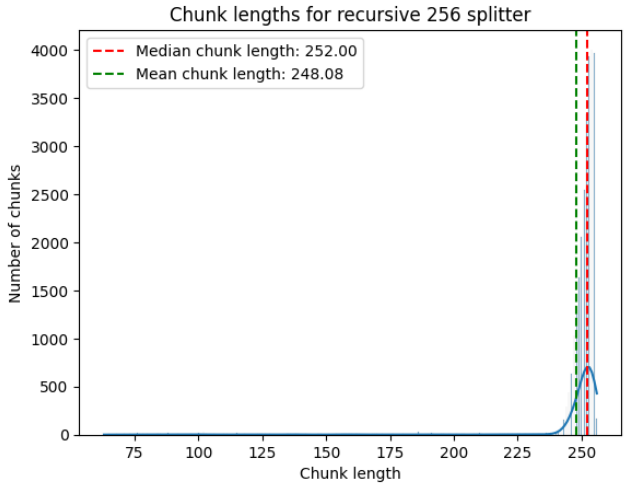
Key Differences:
Chunk Size: The main difference between the strategies is the size of the chunks. Smaller chunks (256 characters) break the text into more granular pieces, while larger chunks (1024 characters) preserve more context in each segment.
Overlap: Some methods, like the recursive and NLTK splitters, maintain an overlap (64-128 characters) between consecutive chunks. This overlap ensures that important context is not lost when moving from one chunk to another.
Semantic vs. Structural Chunking: The semantic splitter uses sentence boundaries to create chunks that are more logically structured and contextually coherent. In contrast, the recursive and NLTK methods use fixed-size chunks, which may split text at arbitrary points, potentially breaking the flow of information across chunk boundaries.
Content Coherence: The semantic splitter creates chunks that are naturally more coherent and meaningful, while the other methods may result in chunks that are less contextually rich but easier to process for specific tasks like tokenization or feature extraction.

Analyzing the Chunk Length Differences
The median and mean chunk lengths in table 4 reveal important insights into how each chunking method handles the division of text:

Splitter	Median Chunk Length	Mean Chunk Length	Number of Chunks
Recursive 256	**252**	**248.08**	**0 - 500**
Recursive 1024	**1020**	**947.19**	**0 - 50**
Semantic	**1031**	**1542.15**	**0 - 700**
NLTK 256	**201**	**204.49**	**0 - 900**
NLTK 1024	**932**	**870.07**	**0 - 500**

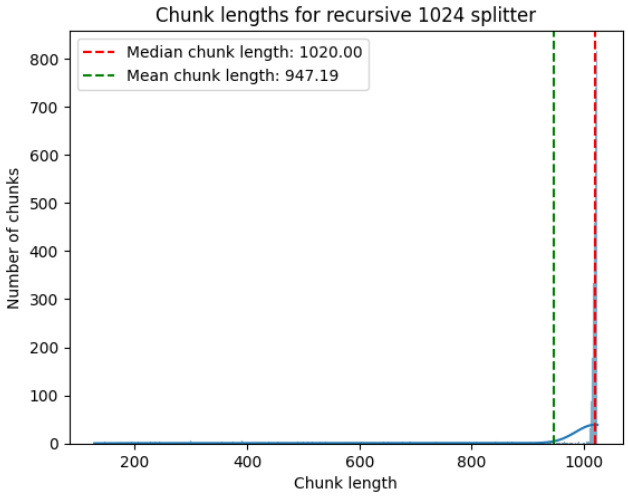
Table 4: Splitter Lengths

Recursive 256: This method produces chunks that are fairly consistent in size, with a median length of 252 and a mean of 248.08. This suggests that the recursive method focuses on keeping chunks close to the defined size of 256 characters, resulting in relatively uniform chunks. The number of chunks varies, ranging between 0 and 500.



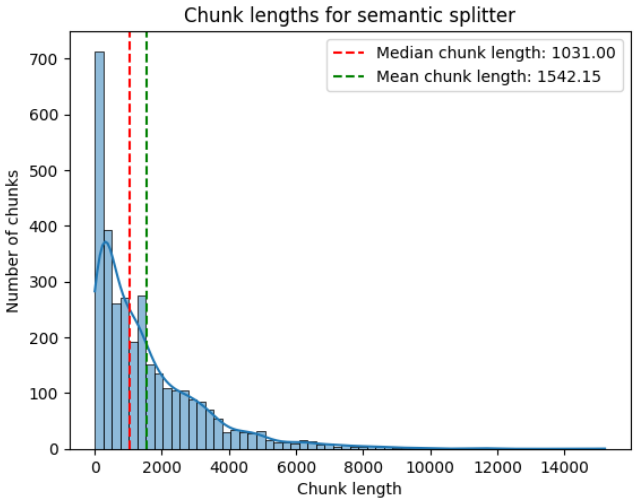
Plot 8: Chunk distribution for recursive 256 splitter

Recursive 1024: The chunks produced by this method have a significantly higher median of 1020 and a mean of 947.19. While the chunks are generally close to the 1024 character target, there is still some variability, likely due to the need to balance chunk size with content boundaries. The number of chunks typically ranges between 0 and 50, indicating fewer chunks compared to smaller chunk sizes.



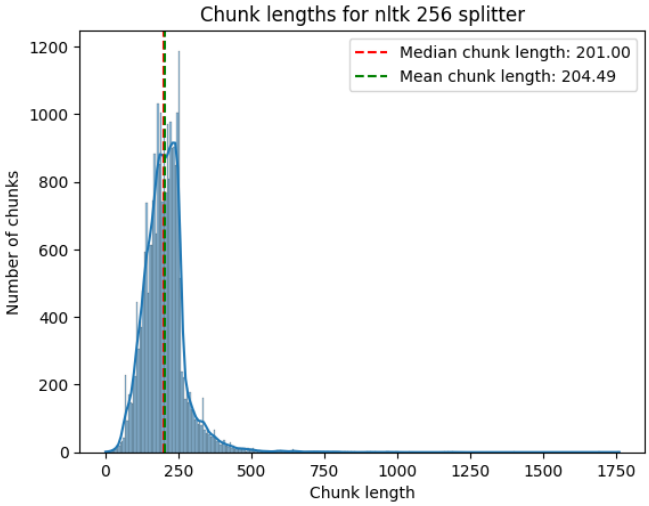
Plot 9: Chunk distribution for recursive 1024 splitter

Semantic: The semantic method has the largest chunks, with a median of 1031 and a mean of 1542.15. The higher mean indicates that many of the chunks are significantly larger than the median, suggesting that the chunking is based on semantic boundaries and may result in fewer, larger chunks, depending on how the content is structured. The number of chunks typically ranges between 0 and 700, with a right-skewed distribution, indicating that the majority of chunk lengths are smaller, while a few are much larger.



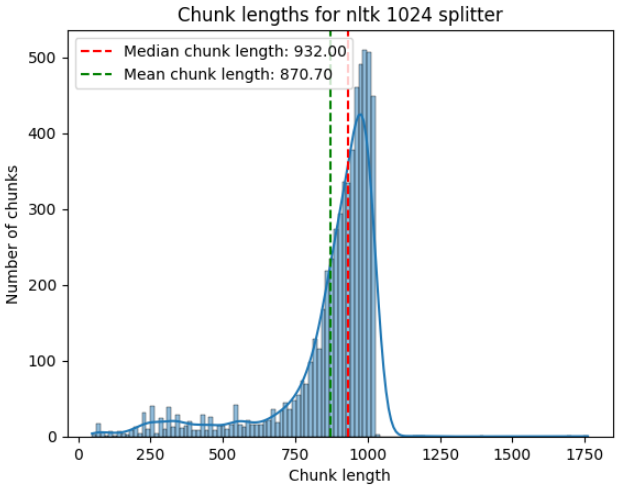
Plot 10: Chunk distribution for semantic splitter

NLTK 256: The chunks here are smaller, with a median length of 201 and a mean of 204.49. The smaller chunks could indicate that NLTK's method breaks the text into smaller, more manageable pieces, likely for more granular processing. Additionally, since it processes entire sentences, sentence lengths may influence the chunk sizes. The number of chunks ranges from 0 to 1000, exhibiting an increased kurtosis resulting in a sharper "bell" curve with a higher peak, highlighting a more concentrated distribution of chunk sizes.



Plot 11: Chunk distribution for nltk 256 splitter

NLTK 1024: Similar to the recursive 1024 method, the chunks produced by NLTK 1024 are quite large, with a median length of 932 and a mean of 870.07. These values suggest that while the chunks are large, they are smaller on average than those created by the semantic chunking method. The number of chunks again ranges between 0 and 500 and is skewed to the left.



Plot 12: Chunk distribution for nltk 1024 splitter

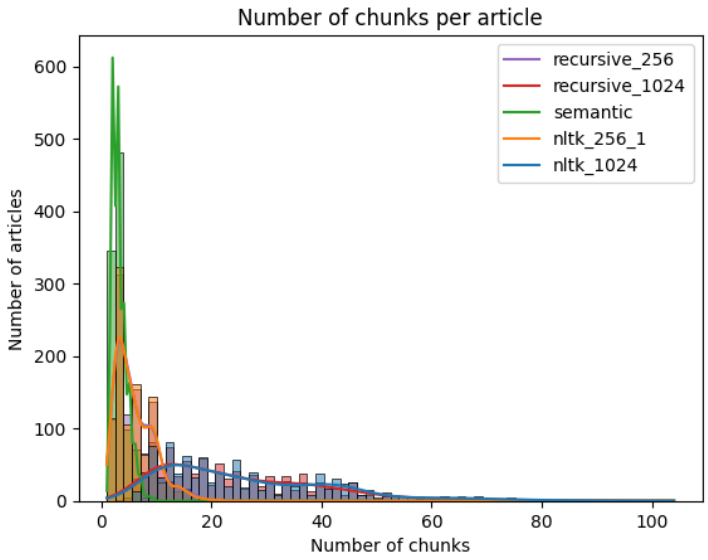
Insights from Chunk Number Distribution

The table 4 above shows the chunking characteristics for different splitters. It appears that the Semantic method has a relatively high median chunk size, but its number of chunks falls in between the NLTK 256 and Recursive 256 methods. Notably, the NLTK 256 method produces the largest number of chunks, while Recursive 256 results in the smallest number. This indicates that the Semantic method generates fewer, larger chunks due to its focus on semantic boundaries.

Additionally, as the chunk size increases, the number of chunks tends to decrease for both the Recursive and NLTK methods. This suggests that larger chunks (as seen in Recursive 1024, Semantic, and NLTK 1024) result in fewer splits, likely due to the method's attempt to maintain chunk size consistency or content structure. Conversely, smaller chunk sizes, such as those seen in NLTK 256, lead to a higher number of chunks, possibly because the method prioritizes breaking the content into smaller, more manageable pieces for finer processing.

The semantic method's chunks are highly variable. This variability is because the method doesn't prioritize chunk size but rather semantic coherence, which leads to more flexible chunking but a wider range of chunk numbers per article.

In some methods, especially the recursive approaches, there is a peak in the distribution indicating that most articles fall within a certain number of chunks, reflecting the tendency of those methods to produce consistently sized chunks.



Plot 13: Combined Chunk distributions

Generating Embeddings

Overview of the Embedding Generation Process

The process of generating embeddings begins after the article chunks have been cleaned and prepared. These chunks are essential for creating a vector store that supports efficient retrieval in the Retrieval-Augmented Generation (RAG) pipeline. Embeddings are generated for each chunk using different models, and the results are indexed in ChromaDB. ChromaDB is a specialized tool for storing high-dimensional data, enabling fast and accurate search functionality, which is vital for the RAG pipeline's information retrieval process.

Integration of Custom Embedding Functions

To generate embeddings, a custom embedding function is created to facilitate the use of various models with ChromaDB. The custom embedding function wraps around different embedding models, ensuring that they are compatible with ChromaDB's indexing system. This function allows for both individual document embedding and batch processing, enabling efficient handling of large datasets and streamlining the embedding process for multiple chunks at once.

Embedding Name	Number of Embeddings
mini_recursive_256	26,630
mini_recursive_1024	6,018
mini_semantic	3,293
mini_nltk_256_1	25,127
mini_nltk_1024	6,059
bge-m3_recursive_256	26,630
bge-m3_recursive_1024	6,018
bge-m3_semantic	3,293
bge-m3_nltk_256_1	25,127
bge-m3_nltk_1024	6,059
gte_recursive_256	26,630
gte_recursive_1024	6,018
gte_semantic	3,293

gte_nltk_256_1	**25,127**
gte_nltk_1024	**6,059**

Table 5: Number of Embeddings

Role of ChromaDB and the Vector Store

ChromaDB is utilized as the vector store for managing the generated embeddings. By converting the article chunks into high-dimensional vectors, ChromaDB makes it possible to index and retrieve data based on semantic similarity. This setup is crucial for enabling fast searches, where similar documents or chunks can be retrieved based on their meaning rather than exact text matches. The embeddings are stored in ChromaDB, allowing the system to perform efficient retrieval operations during the information search phase of the RAG pipeline.

Storing and Managing Embeddings

Embeddings are managed efficiently to ensure the system remains scalable. If embeddings for a particular combination of model and chunk already exist, they are loaded from storage to avoid redundant calculations. If not, new embeddings are generated and saved for future use. This method ensures that the system can scale without the need to repeatedly generate embeddings for previously processed chunks, improving computational efficiency and reducing processing time.

Purpose of Using Multiple Embedding Models

The decision to use multiple embedding models serves to optimize the retrieval process in the RAG pipeline. Each model, such as the mini, bge-m3, and gte models, offers distinct advantages, ranging from faster processing times to potentially higher-quality embeddings. The choice of embedding model impacts the length of the embeddings and their suitability for specific tasks, such as fast retrieval or high-accuracy semantic search. The differences in embedding lengths and model characteristics are important considerations when evaluating the system's performance.

Reflection

The number of embeddings reflects the number of chunks produced by different chunking strategies, rather than the dimensionality of the embeddings themselves. Specifically, smaller chunk sizes, such as 256 characters, typically result in a higher number of chunks, as the text is split into smaller pieces. In contrast, larger chunk sizes, like 1024 characters, lead to fewer chunks because each chunk contains more content. Additionally, chunking strategies based on semantic embeddings tend to produce even fewer chunks. This is because semantic embeddings focus on preserving the meaning and context of the text, which often results in more coherent and larger chunks, reducing the overall count. Therefore, the variation in chunk counts across different strategies can be attributed to the interplay between chunk size, the approach used for chunking, and the underlying content structure.

Storing Embeddings in ChromaDB

To facilitate semantic search and retrieval, embeddings are stored in ChromaDB, a powerful tool designed for indexing and searching high-dimensional data. ChromaDB uses the Hierarchical Navigable Small World (HNSW) algorithm, which is well-regarded for its efficiency in approximate nearest neighbor (ANN) search within high-dimensional spaces. This setup allows for fast and accurate retrieval of relevant documents based on their vector representations.

Vector Spaces and Collections

In ChromaDB, the concept of "collections" is used to store embeddings in separate vector spaces. A collection is akin to an index in a traditional SQL database, but instead of storing rows and columns, it

stores high-dimensional vectors. Each collection is linked to a specific set of embeddings and can be queried to find the most relevant results for a given search query. Collections are essential for organizing embeddings, and they can be easily created or retrieved using ChromaDB's client interface.

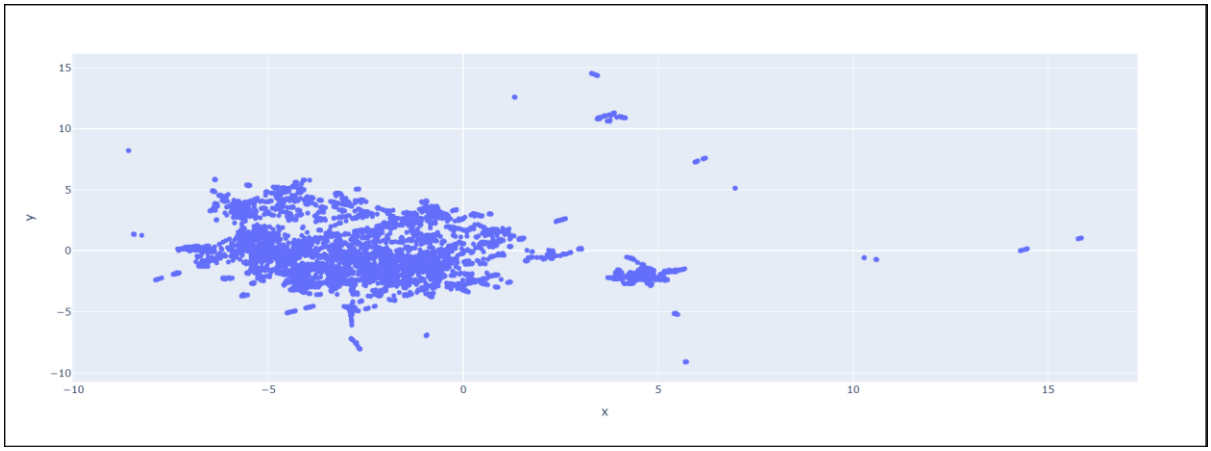
Embedding Storage and Retrieval Process
To store embeddings, a path structure is first created within the data folder to house the embeddings and database files. A ChromaDB client is initialized, connecting to an SQLite server. When creating a collection, the embeddings are stored in batches to optimize the insertion process. Each batch consists of document contents, text embeddings, and metadata, which are added to the collection. The distance metric used to measure similarity between embeddings in the collection is configured to use cosine distance, which is often more effective for semantic similarity compared to the default L2 distance.

Testing the Retrieval Process
Once the embeddings are successfully stored in ChromaDB, it is essential to test the retrieval process. Queries can be run against specific collections to determine how well the system retrieves relevant documents. By querying with a sample text, the system returns the top results that are semantically closest to the input query. The retrieved documents can then be examined to verify if they are contextually appropriate and if the results align with expectations.

Practical Application of ChromaDB for Semantic Search
In a practical scenario, once the embeddings are stored in ChromaDB, they enable the retrieval of highly relevant documents based on the semantic similarity to a given query. This is especially useful for tasks like information retrieval, document summarization, or answering complex queries, where the context and meaning of the input text are critical. By experimenting with various queries, one can assess the system's ability to return meaningful results, ensuring that ChromaDB effectively supports the desired retrieval goals.

Analyzing the Embedding Space Using UMAP
The primary goal of dimensionality reduction in this case is to visualize high-dimensional data in a more interpretable form. By using UMAP, the 768-dimensional embeddings of documents are projected into a 2D space. UMAP preserves both the global structure and the local relationships of the data, allowing us to observe clusters and outliers. The scatter plot of these projected vectors reveals clear clusters of similar documents and highlights the separation between them.

The visualization shows that documents within the same cluster tend to share common characteristics or topics, while those from different clusters exhibit distinct themes. This clustering effect is crucial in understanding the organization of documents within the collection, providing insights into how the retrieval process works based on embedding similarities. Additionally, this 2D representation allows us to zoom into areas of interest and observe the relative positioning of documents, making the retrieval process more intuitive.

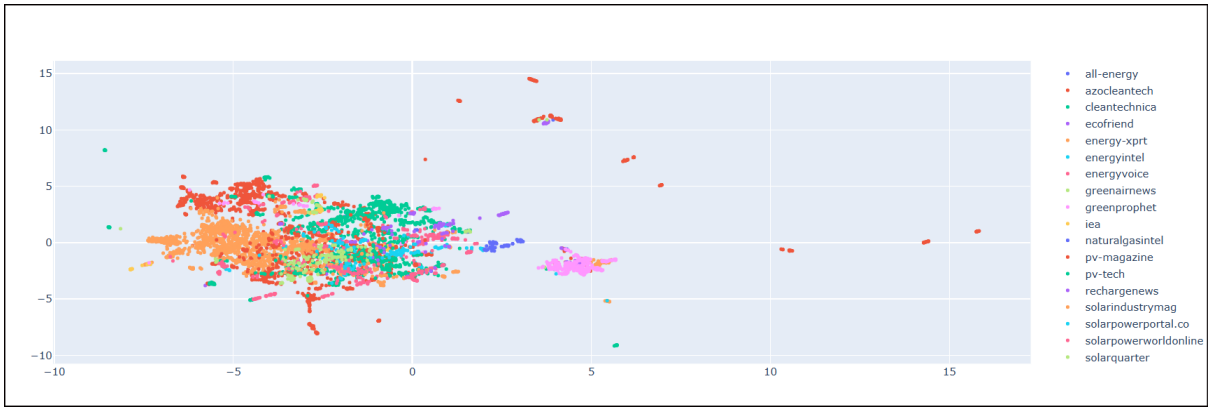


Plot 14: Scatterplot

Clustering and Domain Separation

When we look deeper into the clustering patterns of documents, it becomes evident that documents from the same domain are often grouped together in the embedding space. This suggests that the embeddings capture semantic relationships between documents, effectively clustering them based on their thematic content. The distinct separation between clusters of different domains further emphasizes the effectiveness of the embedding space in distinguishing between diverse topics or categories.

Using UMAP projections, we can observe well-separated clusters corresponding to different domains. This separation indicates that documents from different domains have fundamentally different characteristics. Visualizing these relationships in a lower-dimensional space makes it easier to interpret the boundaries and relationships between different categories, improving the overall understanding of the retrieval process.

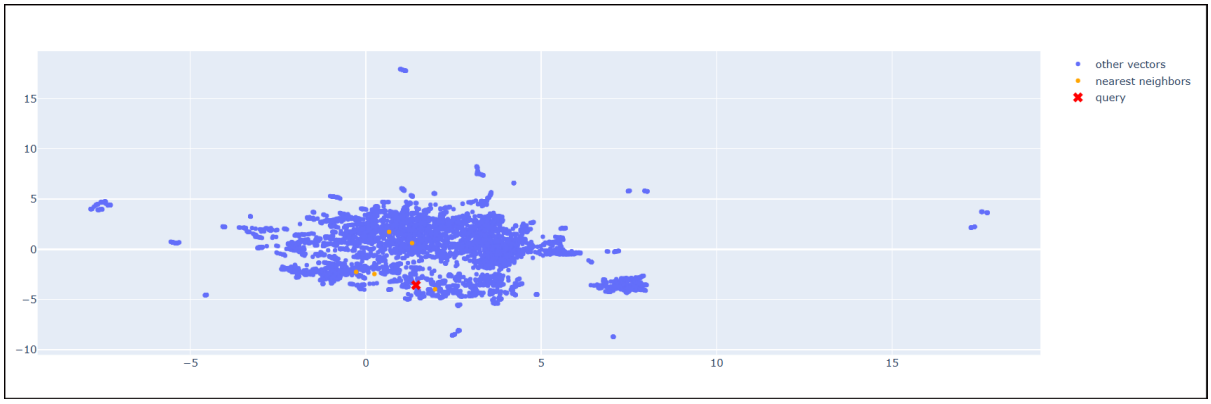


Plot 15: Scatterplot by Domains

Understanding the Retrieval Process through Query Visualization

To better understand how the retrieval process works, it is important to visualize both the query and the most similar chunks in the embedding space. By projecting the query's embedding and the nearest neighbors (similar documents) into the 2D space, we can see how closely related the query is to the retrieved results. The visualization shows that the nearest neighbors tend to form clusters around the query, indicating that the retrieval process is effectively finding the most relevant documents based on their semantic similarity.

In the scatter plot, the query is represented by a red "x", and the most similar documents are marked in orange. The proximity of these orange points to the query highlights the success of the retrieval system in identifying documents with similar semantic content. This visualization allows us to observe the retrieval process's effectiveness, while also acknowledging that the distances in the high-dimensional space might not be fully captured in the 2D projection due to the limitations of dimensionality reduction.

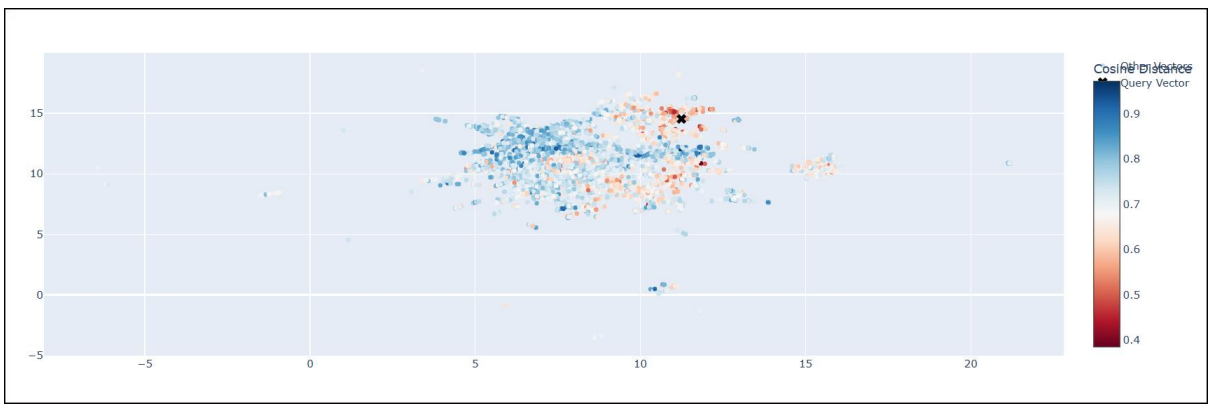


Plot 16: Scatterplot for retrieval process understanding

Cosine Distance and Similarity Analysis

Cosine similarity is a crucial metric for measuring the similarity between document embeddings. It calculates the cosine of the angle between two vectors, with a higher cosine similarity indicating greater similarity between the documents. On the other hand, cosine distance is simply 1 minus the cosine similarity, so smaller distances correspond to higher similarity between document vectors. This measure helps in understanding how closely related the documents are to the query in terms of their semantic content.

In the context of the query retrieval, the cosine distance helps us evaluate the relevance of the retrieved documents. By plotting the cosine distances between the query and various documents in the collection, we can see how documents closer to the query (with smaller cosine distances) are more relevant. This analysis reinforces the concept that the closer a document is to the query in the embedding space, the more semantically similar it is, which directly impacts the effectiveness of the retrieval system. The color-coding in the scatter plot visually represents these distances, providing an intuitive understanding of the results.



Plot 17: Scatterplot for Cosine Similarities

Putting it All Together
To build a functional and accurate RAG (Retrieval-Augmented Generation) pipeline using LangChain and ChromaDB, several steps were taken. Below is a breakdown of what was done, the prompts used, and their evolution to improve model responses.

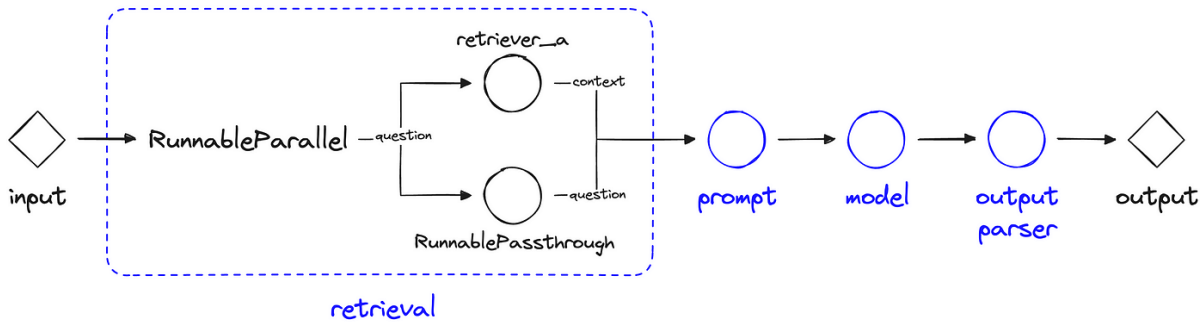
The main objective was to create a system capable of answering user queries based on the information retrieved from a set of documents. This was achieved through the combination of:

Indexing: The article data was first split into smaller chunks. Each chunk was then embedded, and the embeddings were stored in ChromaDB, a vector database, allowing for efficient similarity-based retrieval.

Retrieval: When a user query is input, a vector store retriever is used to find the most relevant chunks based on their similarity to the query.

Generation: After retrieving the relevant chunks, these are passed along with the user query to an LLM (Language Model) to generate a meaningful response. LangChain facilitated the integration of the retrieval and generation processes, ensuring smooth transitions from one to the other.

Langchain
LangChain is an open-source framework that simplifies building applications powered by language models (LLMs). It provides tools for integrating LLMs with external data sources, APIs, and tools, enabling the creation of complex applications such as chatbots, document retrieval systems, and content generation. LangChain streamlines tasks like data retrieval, memory management, and prompt handling, allowing developers to focus on building sophisticated AI-powered solutions without dealing with low-level complexities.
At its core, LangChain enables the orchestration of workflows using **chains** and **agents**, where chains define a sequence of actions, and agents dynamically decide the steps based on inputs. The framework also allows for maintaining context and memory, making it suitable for applications that require awareness of past interactions. By managing prompt templates and handling data retrieval from various sources, LangChain empowers developers to create more accurate, dynamic, and context-aware language model applications.



Plot 18: Framework

The following sections cover the prompts that were tested to refine the generation process.

Prompt 1: Original Template (First Attempt)
The first prompt was a simple structure designed to provide a direct response based on the retrieved context. The focus here was on answering the user's query using the retrieved context with minimal additional processing.

Template:
You are an assistant for question-answering tasks. Use the following pieces of retrieved context to answer the question. If you don't know the answer, just say that you don't know. Keep the answer concise.

Question:
For example, in response to the query, "Where are the biggest increases in wildfire smoke exposure in recent years?", the model returned:
Answer:
"The biggest increases in wildfire smoke exposure in recent years are in the Great Plains region, from North Dakota to Texas."

This answer was brief but correct, providing direct information. However, it lacked nuance, broader context, or explanations of contributing factors such as climate change or fire management practices. While it answered the question, it didn't offer much depth.

Prompt 2: Summary-Based Insight Extraction (Second Attempt)
The second prompt aimed to enhance the quality of responses by encouraging the model to summarize key insights from the retrieved context. The idea was to help the model provide more context and meaningful answers.

Template:
You are a helpful assistant that extracts insights from the retrieved context to answer the user's question. Your task is to summarize the key points from the context and provide the most insightful and relevant answer.
Instructions:
- Provide a brief summary of the context that is most relevant to answering the question.
- Focus on key insights and avoid irrelevant or detailed explanations.
- Keep the answer concise but comprehensive.
- If the context doesn't provide a clear answer, say "I don't know."

Question:
For the same question about wildfire smoke, the response was richer:
Answer:
"The biggest increases in wildfire smoke exposure in recent years are in the Great Plains region, from North Dakota to Texas. This increase is driven by factors such as human activities, including starting the majority of wildfires, climate change causing drier conditions, and a history of aggressive fire suppression."

This answer identified the location and also included contributing factors like human behavior and climate change. It offered more depth and context, making it a more informative response.

Prompt 3: Recent and Relevant Information Prioritization (Third Attempt)
The third prompt refined the model's ability to prioritize the most recent and relevant information. This was done to ensure that the model would focus on the most up-to-date facts, rather than offering information that might be outdated or less applicable to the user's query.

Template:
You are an assistant that answers questions based on context retrieved from multiple sources. Some of the context may be outdated or irrelevant. Your goal is to use the most recent and relevant information to answer the question.
Instructions:
- Prioritize the most recent and relevant information.
- Ignore outdated or irrelevant context that does not help answer the question.
- Provide a concise answer with a focus on the current or most up-to-date knowledge.
- If the answer is not available in the context, say "I don't know."

Question:
For the wildfire smoke question, this prompt focused on providing the latest data:

Answer:
"The biggest increases in wildfire smoke exposure in recent years have been observed in the Great Plains region, primarily from North Dakota to Texas. Factors include human activities such as wildfire ignition and changes in climate patterns."

This answer was focused on current factors and trends, providing a timely response. The model highlighted the critical factors contributing to the changes in smoke exposure, offering a more precise and contextually relevant answer.

Comparison and Insights
First Prompt: The first template provided a simple and straightforward answer but lacked detail or deeper insights. While effective for basic information retrieval, it missed the opportunity to elaborate on the broader context of the issue.

Second Prompt: This template resulted in a much richer response by summarizing and extracting key insights. It expanded on the causes and consequences of the issue, making it more comprehensive. It was ideal for cases where deeper context and nuance were necessary for answering the question.

Third Prompt: The third template focused on the most recent and relevant information, which resulted in a more concise and direct response. While this was useful for minimizing irrelevant context, it potentially left out critical background details that could have provided a more comprehensive answer.

Winner:
The second prompt is likely the most effective for generating rich and insightful answers, as it encourages the model to synthesize the context and deliver a more complete response. However, the third prompt may be more appropriate when time-sensitive or focused answers are required.

Through successive iterations, the prompts evolved from simple, concise answers to more detailed and nuanced insights. Each version progressively improved the model's ability to provide richer,

context-aware answers. The final prompt focused on prioritizing the most relevant and recent information, ensuring that the generated responses were not only accurate but also timely and well-rounded.

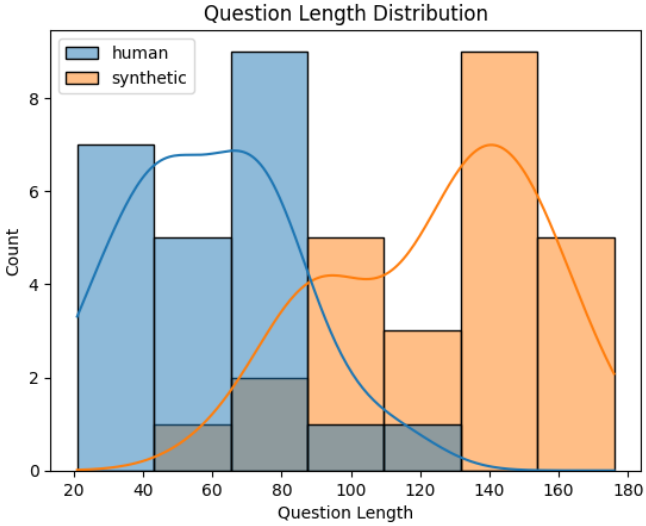
This process of refining prompts is crucial for building an efficient and effective Retrieval-Augmented Generation (RAG) pipeline, ensuring that the system generates responses that are not just relevant but also insightful and comprehensive.

However, for the further evaluation process, only the first instruction (baseline) is considered.

Experimenting with Different k Values in the RAG Pipeline
In the RAG pipeline, the number of chunks retrieved, denoted as **k**, was tested with values of **1**, **3**, and **5**. A smaller k (1) retrieves the most relevant chunk, potentially offering focused but limited answers, while larger values (3 and 5) provide more context, which may lead to broader but potentially less precise responses. By adjusting k for each collection and creating separate QA chains, the impact of different k values on answer relevance and completeness was evaluated and compared to identify the optimal configuration for the pipeline.

Generating Synthetic Question-Answer Pairs
To expand the evaluation question pool, synthetic question-answer pairs were generated using a random selection of documents. The generate_synthetic_qa_pairs function created new questions by prompting the LLM with content from randomly chosen documents and generating answers based on the same content. This approach allowed for the augmentation of the evaluation set, creating a diverse set of evaluation pairs. The synthetic pairs were then stored and processed for use in the evaluation pipeline.

Question Length Distribution and Evaluation Dataset
After generating both human and synthetic evaluation datasets, the distribution of question lengths was analyzed using histograms. The results highlighted the distribution of question lengths across both datasets, revealing insights into the question complexity. The two datasets were then combined into a single evaluation dataframe, with a new column indicating whether a question was synthetic or human-generated. This combined dataset was used for the final evaluation, with 25 synthetic and 23 human-generated questions included.



Plot 19: Question Length Distribution

Note: Issue with Ground Truth Answer Generation
During the evaluation process, an issue arose due to incorrect naming of the ground truth as "answers," which led to an "NA" result for the evaluation. This problem caused delays and took an additional day to resolve, as the ground truth answers needed to be correctly generated before evaluation could proceed. The solution involved using the generate_eval_answers function to ensure the proper generation of answers based on the provided evaluation questions. This step was crucial for the effectiveness of the subsequent evaluation framework.

Impact of Doubling Questions and Answers
Doubling the number of questions and answers increased the dataset size, enriching it with more diverse data. This improvement allows for more robust evaluation of the model's performance. However, it was observed that the length of synthetic questions was generally longer than the original ones. This difference in length could imply that synthetic questions might be somewhat easier to answer. This should be considered in the evaluation process, as it could affect the performance of the RAG pipeline.

RAGAS Metrics
RAGAS provides a suite of metrics designed to evaluate the performance of RAG pipelines. These metrics include:
Answer Relevancy: Measures how relevant the generated answer is to the query by evaluating the cosine similarity between the original question and artificially constructed questions based on the generated answer.
Answer Correctness: Combines semantic similarity and factual accuracy to assess how correct the generated answer is when compared to the ground truth.
Faithfulness: Measures how faithful the generated answer is to the retrieved context. It ensures that all claims made in the generated answer can be verified from the context.
Context Relevancy: Evaluates how relevant the retrieved context is to the query by assessing the proportion of relevant sentences in the retrieved context.

Dataset and Evaluation Pipeline
The evaluation pipeline utilizes a test dataset for each RAG pipeline, which includes the evaluation questions and their corresponding ground truth answers. These questions are processed through the RAG pipeline, generating answers and retrieving relevant context. The RAGAS metrics are then applied to evaluate the pipeline’s performance. This process allows for a comprehensive analysis of the effectiveness of the RAG system.

Results Collection and Visualization
The evaluation results are stored in a structured manner, allowing for easy analysis and comparison over time. Results are visualized using boxplots and bar plots to highlight the distribution and comparative performance across different RAGAS metrics. These visualizations aid in understanding the model’s strengths and weaknesses and can guide further improvements.

Note:
The increased dataset size has bolstered the evaluation process, offering more insights into the RAG pipeline's performance. However, the discrepancy in question length between synthetic and original questions may influence the evaluation metrics. Future improvements could focus on refining the synthetic question generation process to align better with the original question structure to avoid any potential biases in the results.

Initializing LLM Evaluation Results

To evaluate the performance of the large language models (LLMs) across multiple datasets, a dictionary was initialized to store the results for each dataset. The evaluation function was applied to each dataset in the provided list, and the outcomes were stored in the dictionary for further analysis. This process ensured that the evaluation results were organized by dataset name, making it easier to track and compare model performance across different data sources.

Challenges Encountered During Execution

During the evaluation process, multiple technical challenges were encountered that significantly impacted the workflow. After generating a portion of the LLM evaluation results, the Colab environment experienced numerous issues, such as exception errors, CUDA out-of-memory errors, and runtime restarts. These interruptions caused delays and forced the process to restart multiple times. As a result, despite the extended execution time, only a limited number of results (around 10) were successfully generated. The repeated errors and retries led to a total of 7 hours spent solely on resolving these issues.

Loading LLM Evaluation Results

Once the LLM evaluations were completed, the results were stored as CSV files for each dataset. To facilitate the subsequent analysis, a method was implemented to load the evaluation results from these files. The function designed for this purpose systematically checked the availability of the result files for each dataset and, if found, loaded them into memory for further processing. This approach ensured that the data could be accessed easily and in an organized manner.

Dependency Management

Throughout the process, certain dependencies caused conflicts that required additional steps to resolve. To ensure that the necessary libraries for evaluation were available, specific packages, such as evaluate, rouge_score, and bert_score, were re-imported as needed. This was done to ensure that the required functions for computing various evaluation metrics, including ROUGE, Perplexity, and BERTScore, were correctly initialized and available for use in the analysis.

Evaluation Metrics Calculation

The next step in the evaluation process involved computing a set of metrics to assess the performance of the LLMs. These included ROUGE scores (both ROUGE-1 and ROUGE-2), Mean Perplexity, and BERTScore metrics (Precision, Recall, and F1 score). For each dataset, predictions made by the models were compared with the ground truth references to compute these metrics. These scores served as key indicators of model quality, providing insight into how well the LLMs performed in terms of text generation accuracy and relevance.

Plotting and Visualizing Results

After calculating the evaluation metrics, visualizations were created to better understand the performance of the models. Bar plots were generated to illustrate the mean perplexity scores for each model, as well as to compare the scores of various metrics (e.g., ROUGE, Precision, Recall, F1 score) across datasets. These plots provided a clear visual representation of the model's strengths and weaknesses, helping to identify which models performed best under different conditions.

Dataset Evaluation Overview

For k=1:

The evaluation was conducted on a set of four datasets, each identified by a unique name associated with the k1 label. These datasets are:

mini_recursive_256_k1

mini_recursive_1024_k1

mini_semantic_k1

mini_nltk_256_1_k1

The evaluation process involved running the large language models (LLMs) on these datasets and generating a series of performance metrics to assess their effectiveness. The results were then compiled for further analysis, with a focus on various aspects of model performance.

Faithfulness

In terms of faithfulness, the datasets mini_recursive_1024_k1 and mini_semantic_k1 performed the best. These datasets produced responses that were more factually accurate and aligned with the ground truth data. This indicates that the models were better at ensuring the accuracy of their generated text, making them reliable for applications where factual correctness is critical.

Answer Relevancy

When evaluating answer relevancy, the dataset mini_nltk_256_1_k1 emerged as the leader. This dataset produced responses that were most aligned with the given queries, indicating a strong understanding of the user's question. The high score in this category suggests that this dataset is especially adept at generating responses that are contextually appropriate and aligned with the user's intent.

Context Relevancy

For context relevancy, the mini_nltk_256_1_k1 dataset again demonstrated superior performance. This dataset showed a marked improvement in understanding the context of the queries, with the generated responses maintaining coherence and relevance throughout. The strong performance in this category suggests that the model underlying this dataset is particularly good at maintaining context in longer or more complex interactions.

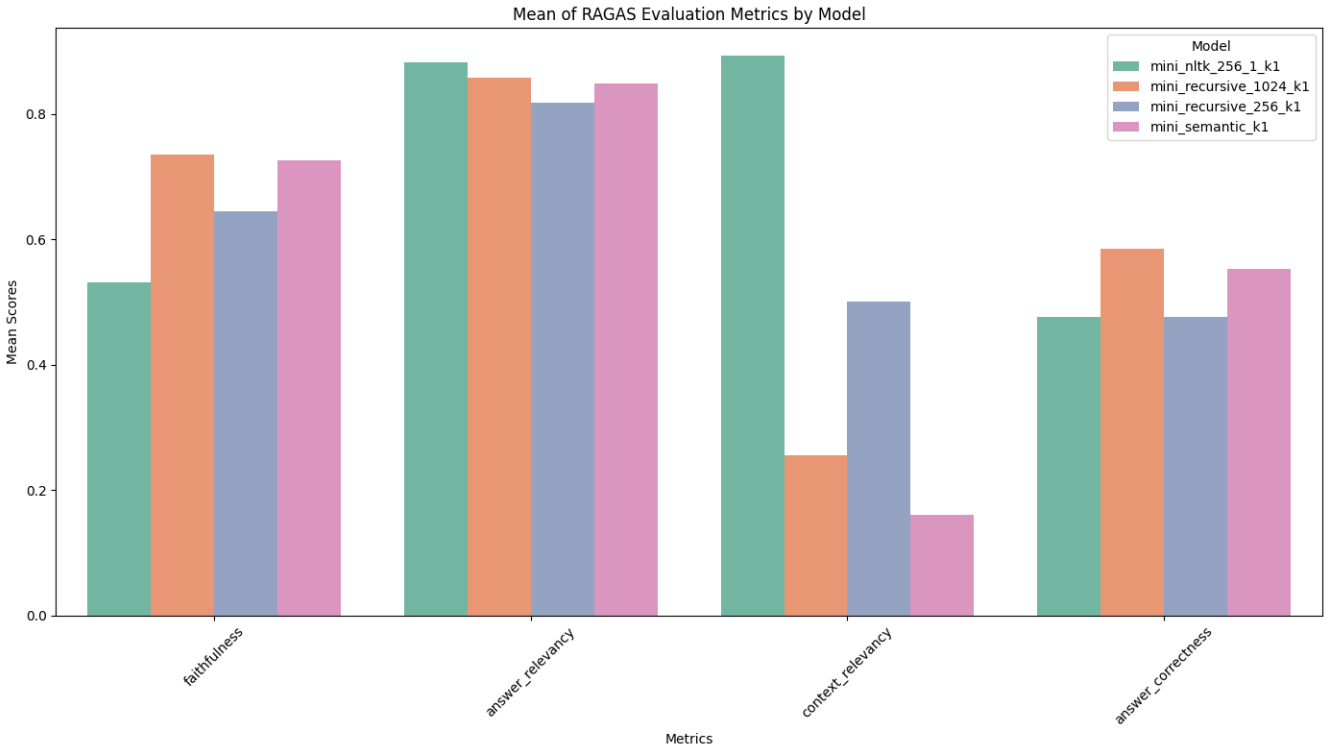
Answer Correctness

Lastly, for answer correctness, both mini_recursive_1024_k1 and mini_semantic_k1 achieved the highest scores. This indicates that these models were more capable of generating accurate and informative answers, which could be particularly valuable for tasks requiring detailed, precise responses. The combination of high faithfulness and answer correctness in these datasets positions them as strong candidates for applications demanding high accuracy.

Overall Insights:

Balanced Accuracy: If accuracy and reliability are key priorities, mini_recursive_1024_k1 and mini_semantic_k1 are the best choices, as they balance both faithfulness and answer correctness.

Contextual Adaptability: For applications that demand an understanding of context and relevancy, especially in multi-turn conversations, mini_nltk_256_1_k1 should be preferred.



Plot 20: RAGAS Metrics for k=1 models

Additional Evaluation Metrics:

ROUGE Scores
ROUGE-1:
mini_recursive_1024_k1: 0.555
mini_semantic_k1: 0.516
mini_nltk_256_1_k1: 0.482
mini_recursive_256_k1: 0.461
Insight: The mini_recursive_1024_k1 dataset achieved the highest ROUGE-1 score, indicating it generated the most relevant unigram overlap with the reference texts.

ROUGE-2:
mini_recursive_1024_k1: 0.431
mini_semantic_k1: 0.383
mini_nltk_256_1_k1: 0.364
mini_recursive_256_k1: 0.333
Insight: Similarly, mini_recursive_1024_k1 achieved the highest ROUGE-2 score, suggesting a stronger performance in bigram overlap and a more detailed understanding of the generated responses.

Precision
mini_recursive_1024_k1: 0.941
mini_nltk_256_1_k1: 0.940
mini_recursive_256_k1: 0.936
mini_semantic_k1: 0.930

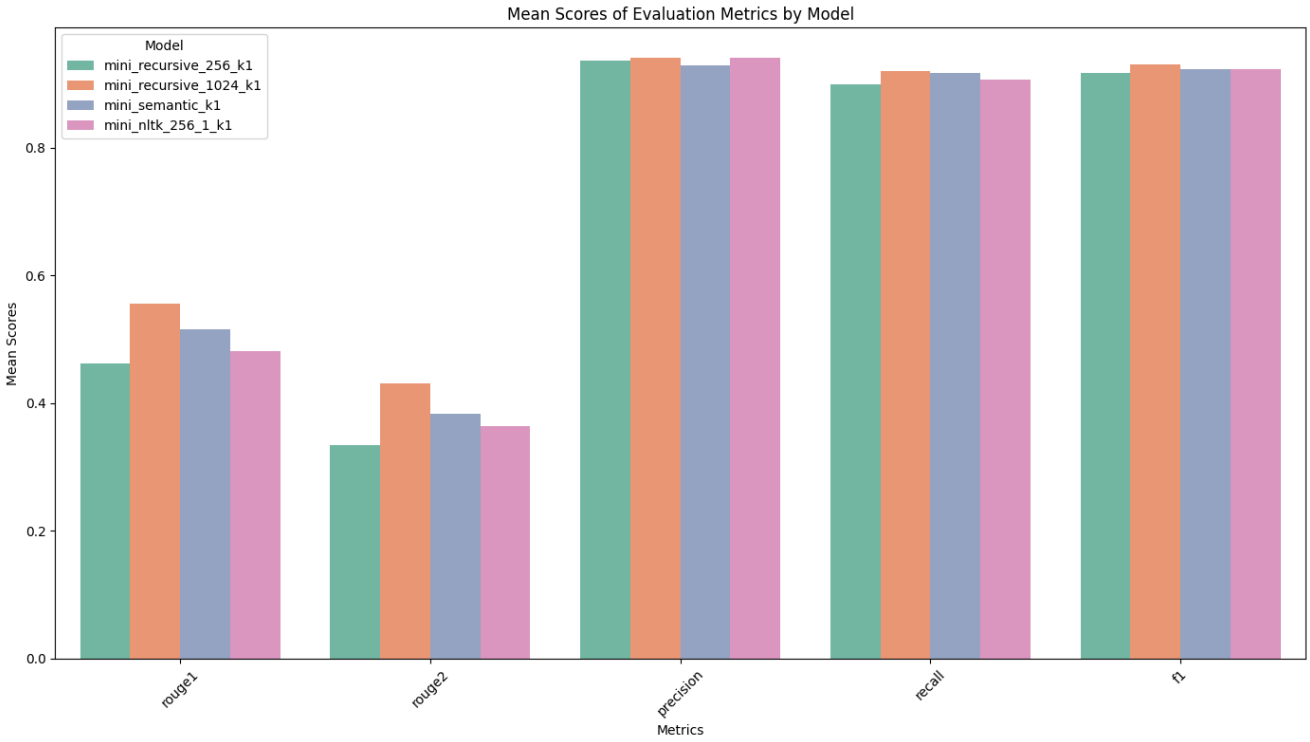
Insight: The precision scores are very close across the datasets, with mini_recursive_1024_k1 slightly leading. This indicates that the models performed similarly in terms of the correctness of the generated responses relative to the total number of responses.
Recall
mini_recursive_1024_k1: 0.920
mini_semantic_k1: 0.917
mini_nltk_256_1_k1: 0.907
mini_recursive_256_k1: 0.899
Insight: mini_recursive_1024_k1 and mini_semantic_k1 achieved the highest recall, meaning these datasets were better at covering the relevant aspects of the reference responses.

F1 Score
mini_recursive_1024_k1: 0.930
mini_semantic_k1: 0.923
mini_nltk_256_1_k1: 0.923
mini_recursive_256_k1: 0.917
Insight: The F1 scores are close, but mini_recursive_1024_k1 leads slightly, indicating a balanced combination of precision and recall. mini_nltk_256_1_k1 and mini_semantic_k1 follow closely behind.

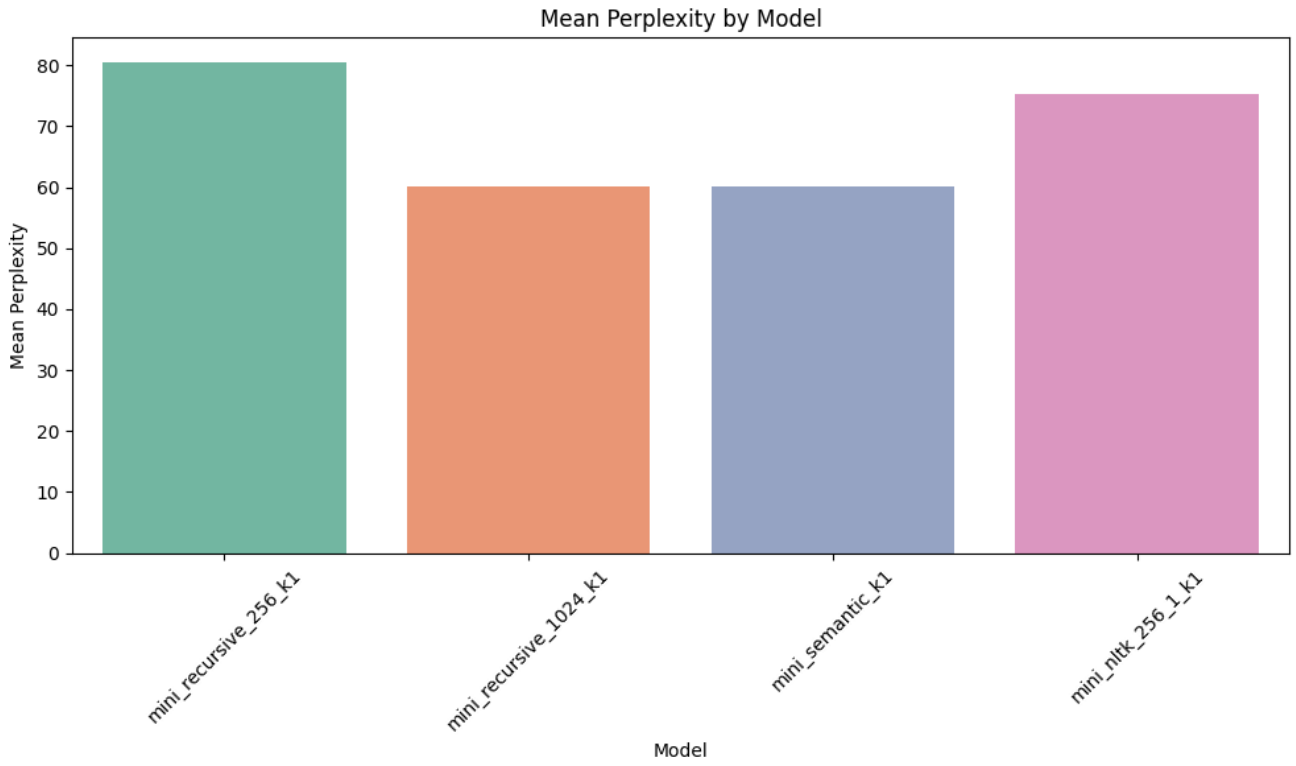
Perplexity
mini_recursive_1024_k1: 60.12
mini_semantic_k1: 60.23
mini_nltk_256_1_k1: 75.21
mini_recursive_256_k1: 80.49
Insight: The mini_recursive_1024_k1 dataset has the lowest perplexity, suggesting that it is more confident and accurate in generating responses. mini_recursive_256_k1 had the highest perplexity, indicating slightly more uncertainty in its generated responses.

Overall Insights
Balanced Performance: The mini_recursive_1024_k1 dataset excelled in terms of ROUGE scores, precision, recall, and F1 score, making it a strong candidate for tasks requiring both precision and recall in generating accurate responses. It also demonstrated the lowest perplexity, indicating higher confidence in its predictions.

Contextual and Precise Responses: The mini_semantic_k1 dataset was closely following in terms of recall, precision, and F1 score. It also performed very well in ROUGE scores, making it a reliable choice for generating semantically accurate responses.



Plot 21: ROUGE and BERT Scores for k=1



Plot 22: Perplexity for k=1

For k =3

The evaluation was conducted on a set of three datasets, each identified by a unique name associated with the k3 label. These datasets are:

mini_recursive_256_k3

mini_recursive_1024_k3
mini_semantic_k3
Unfortunately, mini_nltk_256_k3 and others could not be evaluated due to computing limitations, but the results for the other datasets were assessed and analyzed across key performance metrics, including faithfulness, answer relevancy, context relevancy, and answer correctness.

Faithfulness
In terms of faithfulness, the three datasets — mini_recursive_256_k3, mini_recursive_1024_k3, and mini_semantic_k3 — performed similarly. All three models produced responses that were almost identical in terms of factual accuracy. This indicates that they were equally reliable in generating factually correct responses, making them suitable for applications where faithfulness is a priority.

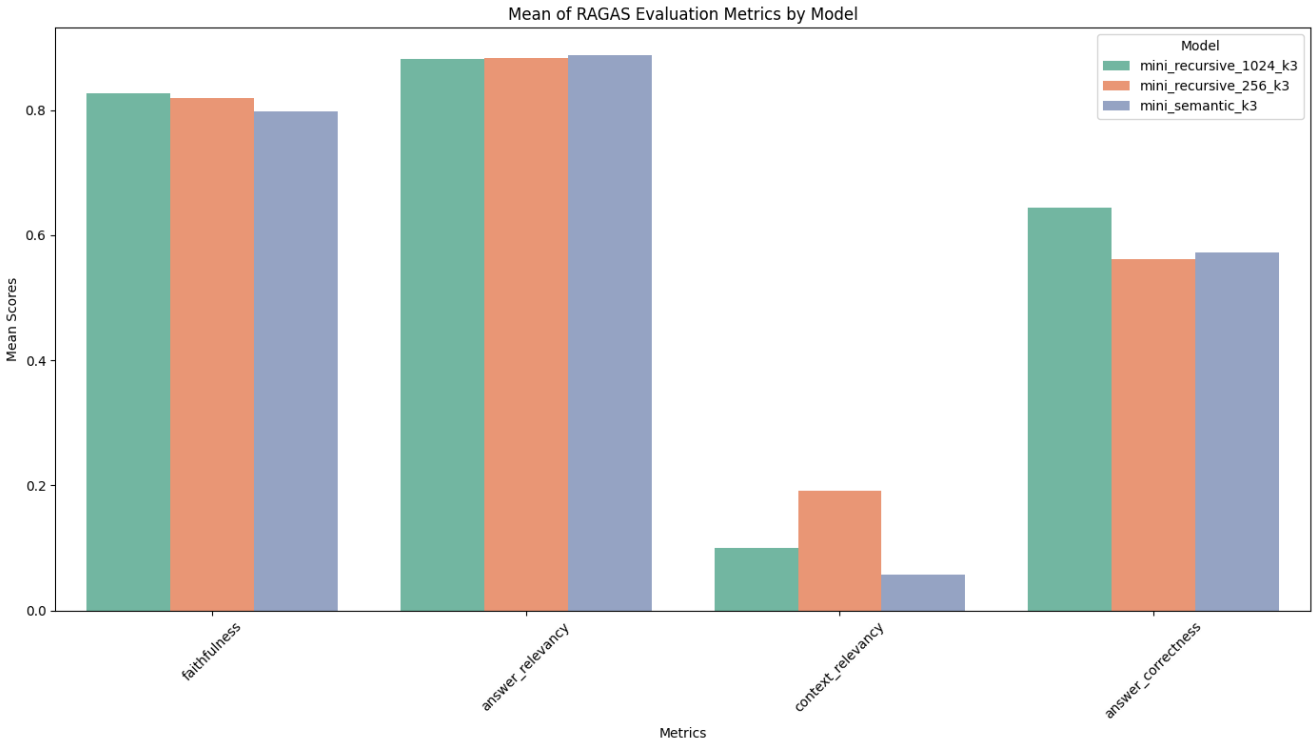
Answer Relevancy
For answer relevancy, there was no significant distinction between the three datasets. mini_recursive_256_k3, mini_recursive_1024_k3, and mini_semantic_k3 all produced responses that were aligned with the given queries, suggesting that all models effectively understood the user's intent and provided relevant answers.

Context Relevancy
When evaluating context relevancy, mini_recursive_256_k3 stood out as the best performer. This dataset demonstrated superior understanding of context, ensuring that the responses were coherent and appropriately aligned with the provided context. The model behind this dataset appears to handle more complex interactions and multi-turn queries effectively.

Answer Correctness
For answer correctness, mini_recursive_256_k3 again showed the highest performance. It consistently produced accurate and informative answers, suggesting that this model excels in providing detailed and precise responses. This makes it a strong candidate for tasks requiring both correctness and contextual understanding.

Overall Insights
Balanced Performance: The three datasets demonstrated similar performance in faithfulness and answer relevancy. mini_recursive_256_k3, mini_recursive_1024_k3, and mini_semantic_k3 all provided reliable and relevant answers, making them suitable for use in applications that prioritize accuracy and relevancy.
Contextual Strength: mini_recursive_256_k3 is the dataset to prefer if the goal is to maintain context in longer or more complex interactions. It excelled in context relevancy and answer correctness, making it particularly well-suited for applications requiring a deeper understanding of context.

Plot 23: RAGAS Metrics for k=3



Additional Evaluation Metrics:
2ROUGE Scores
ROUGE-1:
mini_recursive_1024_k3: 0.574
mini_semantic_k3: 0.536
mini_recursive_256_k3: 0.504
Insight: mini_recursive_1024_k3 achieved the highest ROUGE-1 score, demonstrating the best unigram overlap with reference texts. It outperformed the other models in generating relevant content.

ROUGE-2:
mini_recursive_1024_k3: 0.436
mini_semantic_k3: 0.395
mini_recursive_256_k3: 0.369
Insight: Again, mini_recursive_1024_k3 leads in ROUGE-2, indicating it performs well with bigram overlap, which reflects a more nuanced understanding of the text.

Precision
mini_recursive_1024_k3: 0.938
mini_recursive_256_k3: 0.937
mini_semantic_k3: 0.933
Insight: Precision scores are close across all three datasets, with mini_recursive_1024_k3 slightly outperforming the others. This suggests that all models are fairly precise in generating relevant responses.

Recall
mini_recursive_1024_k3: 0.927

mini_semantic_k3: 0.923
mini_recursive_256_k3: 0.909
Insight: mini_recursive_1024_k3 leads in recall as well, showing its ability to generate a comprehensive set of relevant responses. mini_recursive_256_k3 shows a slightly lower recall.

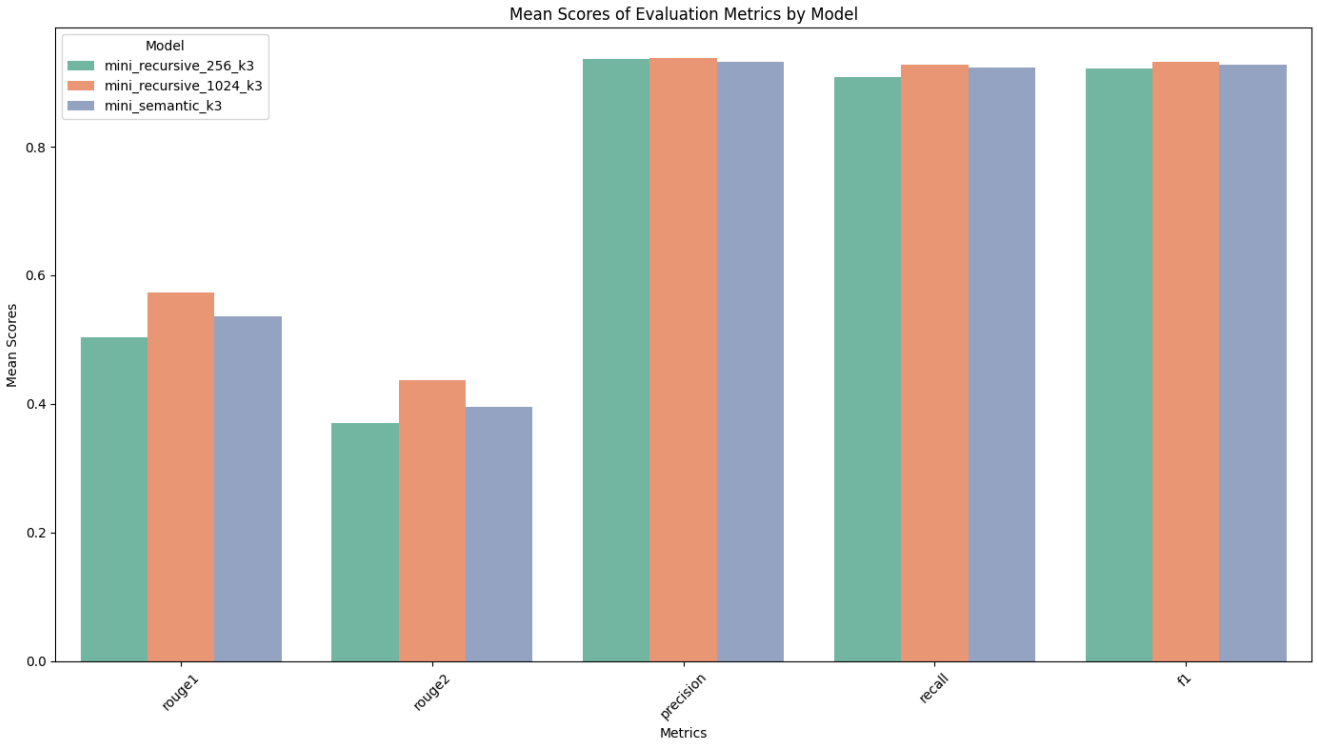
F1 Score
mini_recursive_1024_k3: 0.932
mini_semantic_k3: 0.928
mini_recursive_256_k3: 0.922
Insight: mini_recursive_1024_k3 is also leading in the F1 score, making it the most balanced model when it comes to precision and recall. mini_semantic_k3 follows closely, and mini_recursive_256_k3 performs slightly lower.

Perplexity
mini_recursive_1024_k3: 52.19
mini_semantic_k3: 52.35
mini_recursive_256_k3: 77.09
Insight: mini_recursive_1024_k3 and mini_semantic_k3 have similar perplexity values, indicating low uncertainty in their predictions. mini_recursive_256_k3 shows higher perplexity, indicating more uncertainty in its generated outputs.

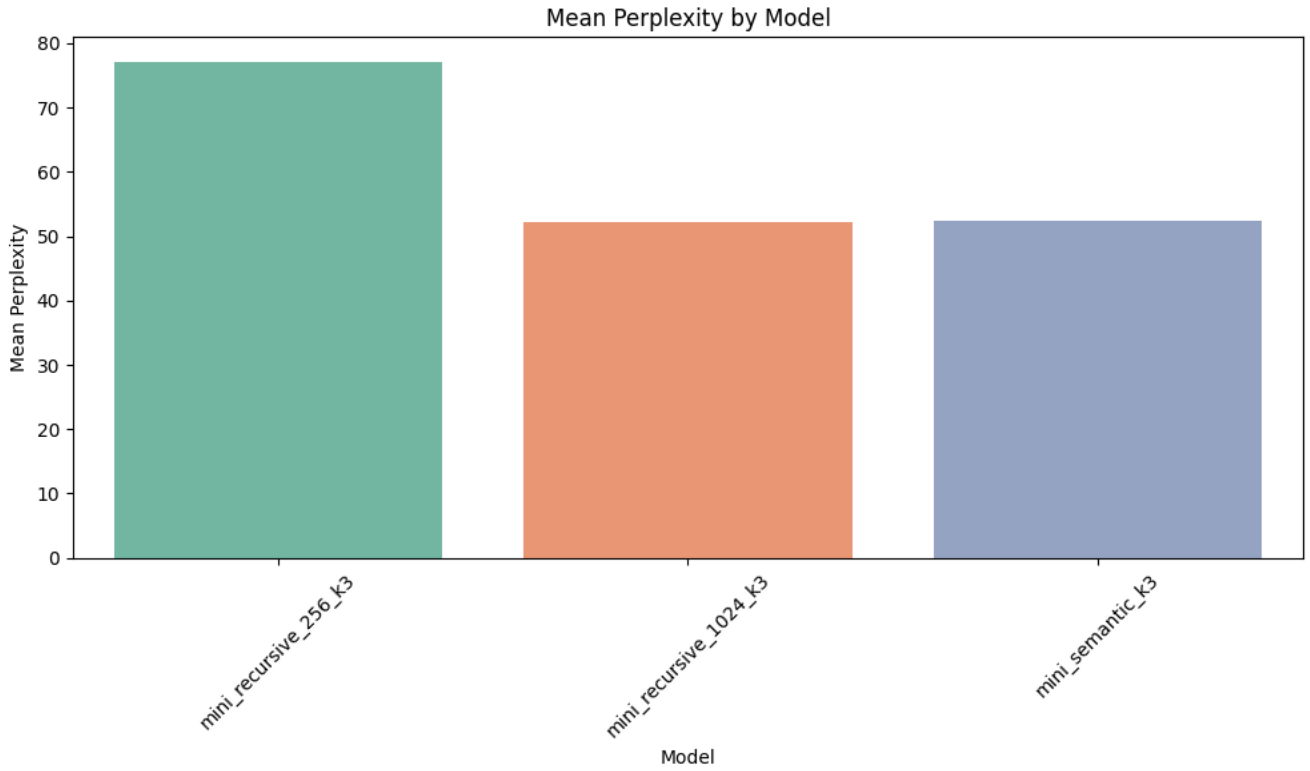
Overall Insights
Top Performer: mini_recursive_1024_k3 excels in almost all metrics, including ROUGE scores, precision, recall, and F1 score. It also has a lower perplexity, suggesting that it is more confident in generating responses.

Strong Competition: mini_semantic_k3 performs closely behind in most metrics, especially in precision, recall, and F1 score. It also shows a low perplexity, which indicates that it generates reliable responses.

Room for Improvement: mini_recursive_256_k3 performs well but has higher perplexity compared to the other two datasets. This suggests that while it provides relevant responses, there is some uncertainty or variability in the output.



Plot 24: ROUGE and BERT Scores for k=3



Plot 25: Perplexity for k = 3

For K=5:

The evaluation was conducted on a set of three datasets, each identified by a unique name associated with the k5 label. These datasets are:

mini_recursive_256_k5

mini_recursive_1024_k5
mini_semantic_k5
Unfortunately, several other datasets could not be evaluated due to computing limitations, including mini_nltk_256_k5, mini_nltk_1024_k5, bge-m3_recursive_256_k5, and bge-m3_recursive_1024_k5. However, the results for the available datasets were assessed across key performance metrics: faithfulness, answer relevancy, context relevancy, and answer correctness.

Faithfulness
In terms of faithfulness, mini_recursive_1024_k5 and mini_semantic_k5 performed the best. Both datasets produced responses that were more accurate and aligned with the ground truth data, ensuring factual correctness. This indicates that these models are reliable for generating responses where accuracy is essential.

Answer Relevancy
When evaluating answer relevancy, the mini_recursive models outperformed mini_semantic_k5. Both mini_recursive_256_k5 and mini_recursive_1024_k5 provided answers that were more relevant to the given queries, suggesting that these models are better at understanding and addressing the user's intent in comparison to the mini_semantic_k5 dataset.

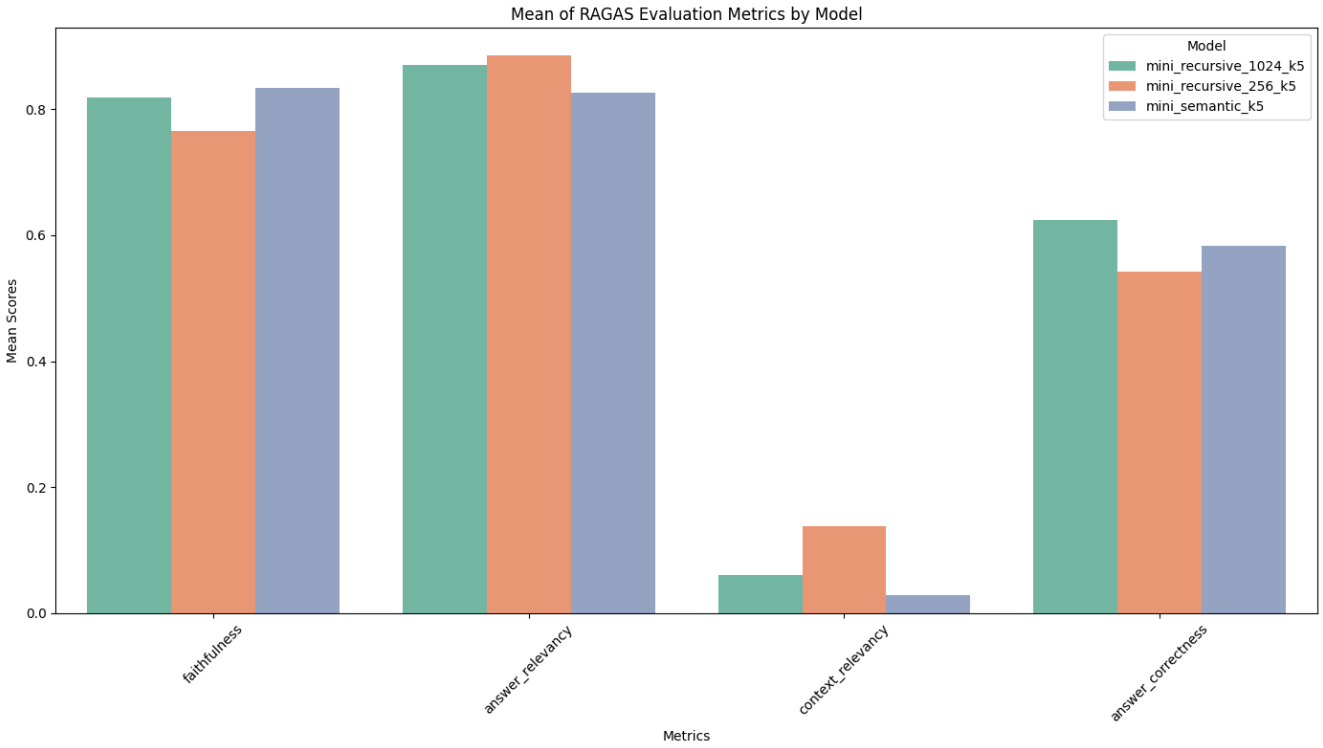
Context Relevancy
For context relevancy, mini_recursive_256_k5 performed the best. This dataset demonstrated a stronger ability to understand the context of the queries, ensuring that responses maintained coherence and were relevant to the ongoing conversation. This suggests that mini_recursive_256_k5 is particularly effective in handling more complex or multi-turn interactions.

Answer Correctness
Lastly, for answer correctness, mini_recursive_1024_k5 and mini_semantic_k5 showed the highest performance. Both datasets produced accurate and informative answers, making them suitable for tasks that require detailed and correct responses. These models excel in ensuring the quality and precision of the answers they generate.

Overall Insights
Balanced Performance: mini_recursive_1024_k5 and mini_semantic_k5 provided the best performance in terms of faithfulness and answer correctness, making them reliable choices for tasks that require factual accuracy and precise answers.

Contextual Understanding: mini_recursive_256_k5 stood out for its strong context relevancy, making it the best choice for applications that need to maintain context over multiple interactions or longer dialogues.

Answer Relevancy: The mini_recursive models overall outperformed mini_semantic_k5 when it comes to understanding the query and providing contextually relevant answers



Plot 26: RAGAS Metrics for k=5

Additional Mertics:
ROUGE Scores
ROUGE-1:
mini_recursive_1024_k5: 0.575
mini_semantic_k5: 0.561
mini_recursive_256_k5: 0.506
Insight: mini_recursive_1024_k5 achieved the highest ROUGE-1 score, indicating the best unigram overlap, followed closely by mini_semantic_k5. mini_recursive_256_k5 showed a lower ROUGE-1 score.

ROUGE-2:
mini_recursive_1024_k5: 0.446
mini_semantic_k5: 0.424
mini_recursive_256_k5: 0.370
Insight: mini_recursive_1024_k5 again leads in ROUGE-2, demonstrating the best bigram overlap, which suggests a better understanding of the text structure.

Precision
mini_recursive_1024_k5: 0.940
mini_semantic_k5: 0.933
mini_recursive_256_k5: 0.934
Insight: mini_recursive_1024_k5 achieves the highest precision, showing that it generates highly relevant responses. mini_recursive_256_k5 and mini_semantic_k5 perform similarly in terms of precision.

Recall
mini_recursive_1024_k5: 0.928
mini_semantic_k5: 0.927
mini_recursive_256_k5: 0.911
Insight: mini_recursive_1024_k5 leads in recall, demonstrating that it retrieves a greater portion of relevant responses. mini_recursive_256_k5 shows a slightly lower recall compared to the other models.

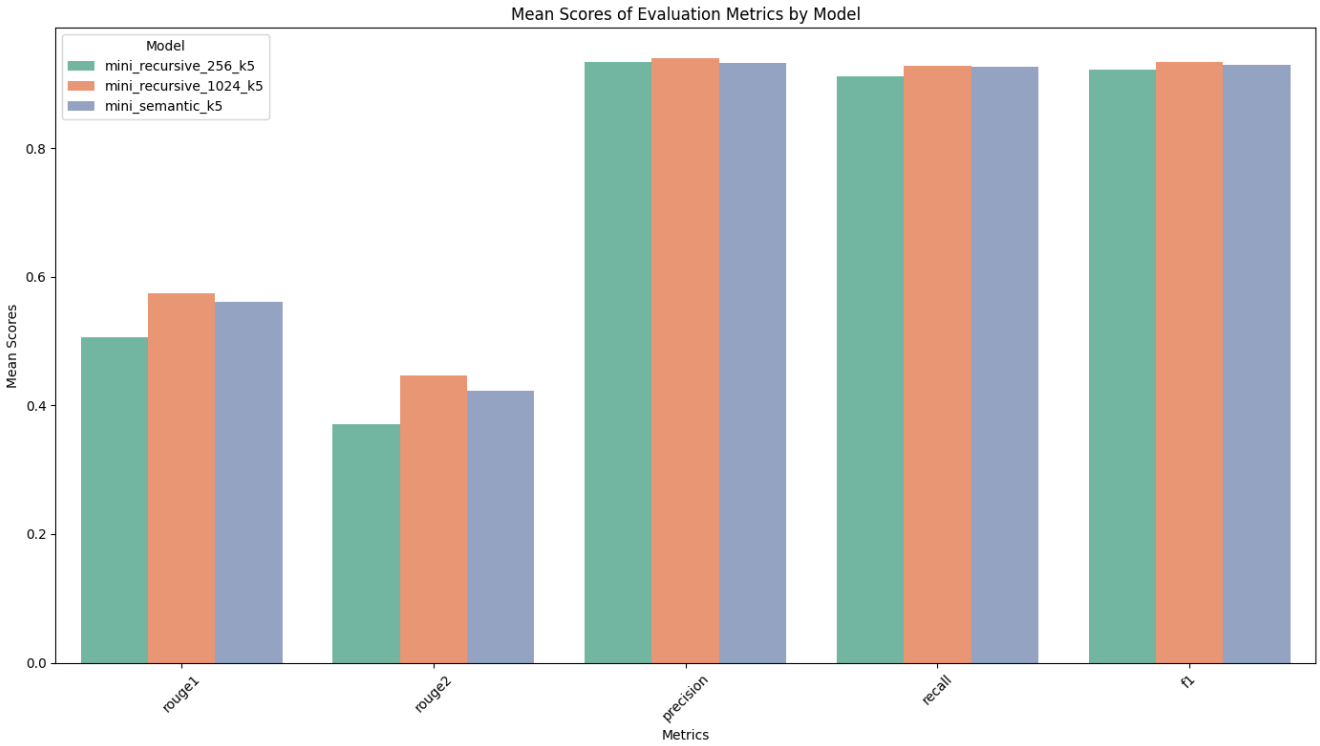
F1 Score
mini_recursive_1024_k5: 0.934
mini_semantic_k5: 0.930
mini_recursive_256_k5: 0.922
Insight: mini_recursive_1024_k5 performs best in F1 score, showing a good balance between precision and recall. mini_semantic_k5 follows closely behind.

Perplexity
mini_recursive_1024_k5: 53.84
mini_semantic_k5: 49.58
mini_recursive_256_k5: 64.61
Insight: mini_semantic_k5 shows the lowest perplexity, suggesting that it is more confident in generating responses. mini_recursive_256_k5 has the highest perplexity, indicating higher uncertainty in its predictions.

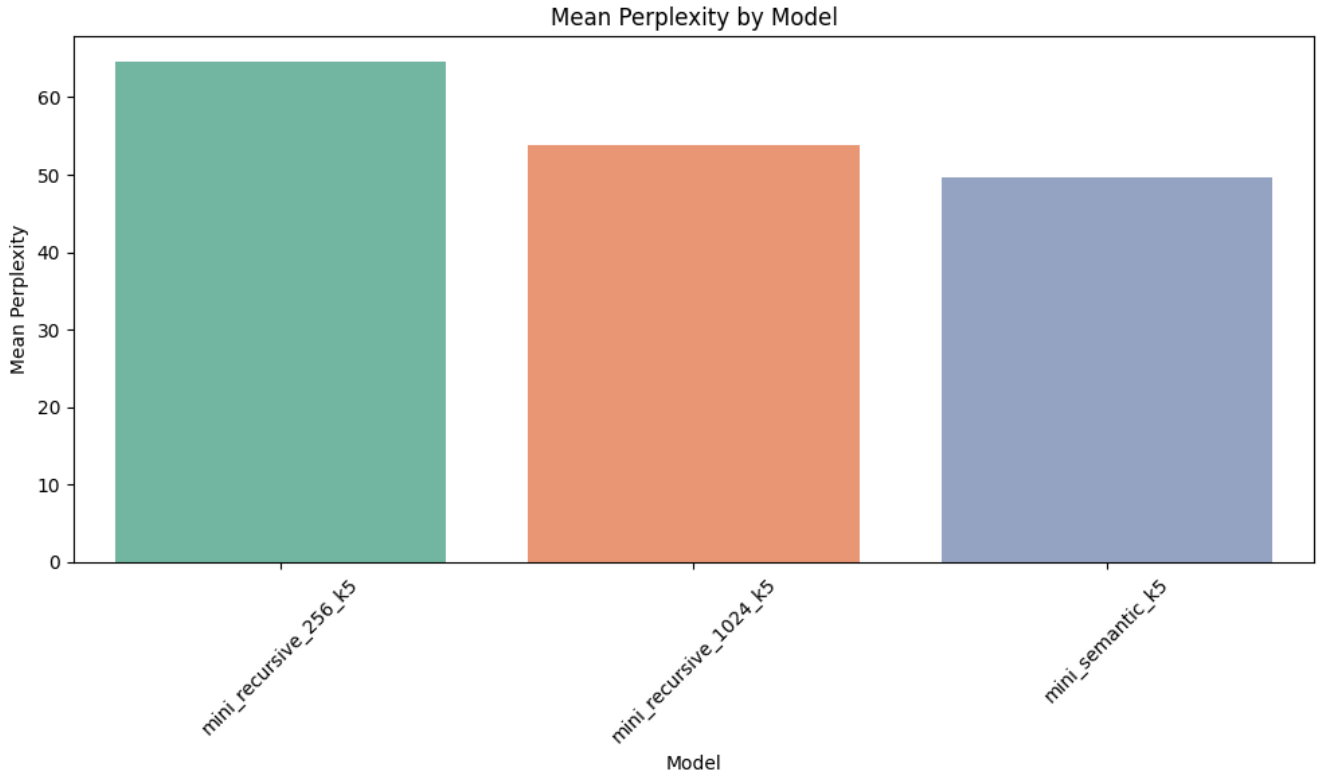
Overall Insights
Top Performer: mini_recursive_1024_k5 outperforms the other models in ROUGE scores, precision, recall, and F1 score. It also performs well in perplexity, suggesting it generates more confident and relevant responses.

Strong Competitor: mini_semantic_k5 shows strong performance, especially in precision, recall, and F1 score. It has the lowest perplexity, indicating that it generates responses with higher certainty.

Room for Improvement: mini_recursive_256_k5 has slightly lower performance across the board, especially in ROUGE scores, recall, and perplexity, but it still generates valuable results with a reasonable precision.



Plot 27: ROUGE and BERT Scores for k=5



Plot 28: Perplexity for k =5

Summary

mini_recursive_1024_k5 stands out as the best-performing model, excelling across all metrics, particularly in ROUGE-1, ROUGE-2, precision, recall, and F1 score. mini_semantic_k5 is a close

competitor, performing well with a low perplexity. mini_recursive_256_k5, while still performing decently, has the highest perplexity and slightly lower ROUGE scores, indicating some areas for improvement.

Overall, these interpretations and comparisons are a good start. However, if other models could have been run, it could have given a much wider scope of understanding

Qualitative Assesment

For the qualitative evaluation, I inspected a few examples from the dataset using the model's output in the mini_nltk_256_k1 to analyze its performance. For instance, I examined the response to the input question: "What is the innovation behind Leclanché's new method to produce lithium-ion batteries?". The answer provided by the model was:

"Leclanché's new method to produce lithium-ion batteries involves using high-capacity NMCA cathodes with a water-based production process, increasing energy density by 20% and reducing cobalt usage."

Upon comparing it to the ground truth:

["The innovation behind Leclanché's new method to produce lithium-ion batteries is replacing highly toxic organic solvents with a water-based process to make nickel-manganese-cobalt-aluminium cathodes (NMCA)."]

The model's output demonstrated a clear and relevant response, capturing the key concepts related to energy density and cobalt reduction. It provided useful technical details in a concise manner. However, there was a slight mismatch in the phrasing. The model mentioned "high-capacity NMCA cathodes" but did not specifically reference the replacement of highly toxic organic solvents with a water-based process, which is the core innovation. This omission slightly reduced the precision of the answer, though the broader concept of using a more sustainable production process was still conveyed.

Similarly, for the question "What is the EU's Green Deal Industrial Plan?", the model output was:

"The EU's Green Deal Industrial Plan aims to boost investment and economic resilience in the bloc to tackle the challenges of the green transition by making its net-zero industry more competitive and accelerating its transition to net zero. It includes strategies such as tax exemptions, flexible aid, and the promotion of local manufacturing."

This response effectively summarized the core objectives of the Green Deal, focusing on the promotion of local manufacturing and competitive strategies. However, it slightly deviated from the ground truth:

["The EU's Green Deal Industrial Plan is a strategy aimed at making its net-zero industry more competitive and speeding up its transition to net zero. It aims to boost European manufacturing of technologies, goods, and services necessary to reach its climate goals."]

While the model's response covers many important aspects, it lacks mention of the broader aim of boosting European manufacturing for climate goals, which could have further clarified the context of the plan.

Overall, from my inspection of these examples, the model provided contextually relevant answers. It was able to distill complex information into coherent summaries but exhibited slight differences in phrasing and the inclusion of finer details. The model demonstrated an understanding of the key points but could be further refined to ensure more precise alignment with the ground truth. Further attention to detail, especially in how key innovations and goals are phrased, would improve the model's performance.

Conclusion

In conclusion, this project successfully demonstrated the application of advanced Natural Language Processing (NLP) techniques through the development of a Retrieval-Augmented Generation (RAG) system tailored for the cleantech sector. By leveraging the Cleantech Media Dataset, which encompasses over 20,000 articles, we were able to extract meaningful insights and enhance our understanding of clean technology innovations. The project involved a comprehensive exploration of various methodologies, including data preprocessing, tokenization, and the implementation of multiple text-splitting techniques. Through rigorous experimentation with different embedding models and RAG prompt templates, we optimized the retrieval process to improve the quality of information generated.

The evaluation framework established for this project provided a robust mechanism for assessing the system's performance using a combination of quantitative metrics and qualitative analysis. The insights gained from this evaluation highlighted the effectiveness of the RAG system in summarizing and retrieving relevant information from a vast corpus of cleantech articles. Despite the challenges encountered—such as computational demands, financial costs associated with API usage, and the intricacies of managing large datasets—the experience was immensely rewarding. The cognitive demands of this project pushed me to my limits but ultimately deepened my understanding of NLP and AI applications in real-world contexts.

Overall Reflection on the NLP Course

Throughout the last 10 weeks of the Natural Language Processing (NLP) course, I have gained invaluable insights and skills that have significantly enhanced my understanding of AI and its applications. The curriculum covered a wide range of topics, including vector databases, embeddings, n-grams, recurrent neural networks (RNNs), transformers, and ultimately Retrieval-Augmented Generation (RAG). Each of these components contributed to a comprehensive learning experience that prepared me for real-world challenges in NLP.

One of the most rewarding aspects of this course was the supportive and understanding environment fostered by Dr. Tomuro. Her guidance made me feel comfortable navigating the complexities of AI, despite being new to the field. This encouragement was crucial in helping me adapt to the challenging material and fostering a sense of confidence in my abilities.

The course was intense, requiring significant time and effort to grasp the intricate concepts presented. I found myself dedicating long hours to assignments and projects, often working late into the night. This rigorous schedule pushed me to enhance my problem-solving skills and resilience. The final project, which involved developing a RAG system using the Cleantech Media Dataset, was particularly demanding but also incredibly fulfilling.

I learned firsthand how critical it is to pay attention to detail in NLP tasks. An error in handling the evaluation dataset cost me an extra day and additional computational resources, highlighting how even small mistakes can have significant repercussions. This experience underscored the importance of thoroughness in data preparation and model evaluation.

Despite the challenges, I genuinely enjoyed the learning process. The course provided a rich blend of theoretical knowledge and practical application, allowing me to engage deeply with the material. I appreciated the opportunity to explore various NLP techniques and their implications for real-world problems, particularly in the cleantech sector.

Overall, this course has been transformative in shaping my understanding of NLP and AI. I am grateful for the support from my professor and peers throughout this journey. The skills and knowledge I have gained will undoubtedly serve as a strong foundation for my future endeavors in artificial intelligence research and application.