Query Focused Multi perspective Answer Summarization

Advanced NLP Project – Project Interim Progress

https://github.com/s-gunalan/query-focused-answersumm/

Gunalan S (2023801009) | Shah Neil Kumar (2022801009) | Radhakrishna Bollineni (2023900016)

1. Abstract

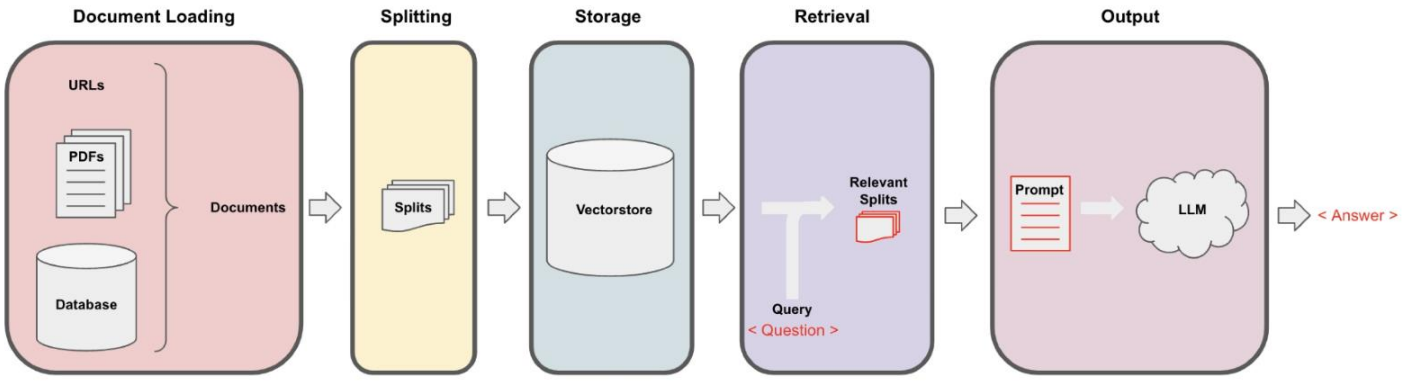
Community Question Answering Forums like Quora, Stack overflow, etc. Answers contain a rich resource of answers to a wide range of questions. Each question thread can receive many answers with different perspectives. Query-guided refers to a technique or approach in natural language processing (NLP) and information retrieval where a user's query or input is used to guide and inform various stages of a computational process, such as search, summarization, or information extraction. In summarization tasks, query-guided models generate summaries that are related to the user's query. This means that the generated summary is tailored to address the specific aspects of the query, ensuring that the most relevant information is included.

1.1. Proposed Methodology:

Two-step approaches in query-focused summarization involve two key components: an extractor model and an abstractor model. The extractor's role is to identify relevant segments of the source document based on the input query, while the abstractor synthesizes these segments into the final summary.

Query-guided multi-perspective answer summarization using two-staged models is a sophisticated approach to generating concise and informative summaries of text in response to a specific query. This technique involves two main stages:

- **Information Retrieval Stage:** In this stage, a model is responsible for selecting relevant documents or passages from a larger corpus of text based on the input query. Common methods used for this stage include Map Reduce, Map Reduce with overlapping chunks, Map Reduce with Rolling Summary and more recent methods using Vector stores (like Vertex Matching Engine, Weaviate, etc.).
- **Summarization Stage:** In this stage, the selected documents or passages are passed to a summarization model, which generates concise and coherent summaries. The summarization model can be based on various architectures, such as Transformer-based models (e.g., PaLM v2, GPT-3, LLAMA, or more recent variants). The input query plays a crucial role here, guiding the model to produce a summary that directly addresses the user's query.



2. Progress

This documentation provides an progress of the two-staged query focused multi-perspective answer summarization system. It includes details on the QA (Question Answering) service and the Auth (Authentication) service, explaining their functionalities, components, and how they interact.

2.1. QA Service (model/qsumm.py)

The QA Service is a critical component of the two-staged question answering system. It is responsible for processing user queries, accessing a document database via the Google Vertex API, and providing answers to questions.

2.1.1. Components

- **Initialization:** Initializes the necessary components and variables for the QA service.
- **Formatter Function:** Defines a formatter function to format and display the results generated by the QA service. This function enhances the readability of the answers provided.
- **ask() Function:** Defines an ask() function that acts as the core of the QA service. It takes user questions as input and retrieves relevant answers from the document database.
- **main() Function:** Defines a main() function that orchestrates the QA service. It starts the service and facilitates user interaction by asking questions and displaying answers.

2.1.2. Functionality

The QA Service enables users to input questions and receive accurate answers from a document database. It leverages natural language processing techniques and the Google Vertex API to perform question answering.

2.2. Auth Service (auth/oauth.py)

The Auth Service plays a crucial role in the project by handling authentication for accessing the Google Vertex API. Authentication is required to obtain an access token, which is necessary for making authorized API calls.

2.2.1. Components

- **Usage of Auth.py:** The Auth Service utilizes the auth.py file to manage authentication processes.
- **Access Token Retrieval:** Obtains an access token using the authentication mechanisms implemented in auth.py before making API calls to the Google Vertex API.

2.2.2. Functionality

The Auth Service ensures secure access to the Google Vertex API by obtaining and managing access tokens. This authentication layer is essential for the QA Service to access the document database.

2.3. Interaction Between Services

- The QA Service and Auth Service are tightly integrated. Before the QA Service can access the document database via the Google Vertex API, it relies on the Auth Service to obtain a valid access token.
- The Auth Service acts as a secure gateway, authenticating the QA Service and allowing it to make authorized API calls to retrieve answers to user queries.

2.3.1. Usage

- Start the QA Service by invoking the main() function.
- Input questions using the ask() function to obtain answers from the document database.
- The Auth Service runs seamlessly in the background, ensuring secure authentication for API access.

2.3.2. Dependencies

- Python
- Google Vertex API
- auth.py (Authentication script)

3. Next Steps:

We are going to explore and select a small-sized Language Model (LLM) for optimal performance, prepare datasets, integrate, and test the QA and Auth services, and continuously optimize the system. Additionally, we plan to develop a user-friendly web UI for interaction and conclude with a presentation of key findings and project outcomes

3.1. Exploration of LLM:

- Explore LLM of small sized LLM with comparable performance
- Compare and evaluate the LLMs performance and choose the optimal LLM for the project

3.2. Dataset Preparation and Preprocessing:

- Load the QMSum Dataset and AquaMuse Dataset as discussed in the research papers.
- Preprocess the dataset as needed, including text cleaning, tokenization, and formatting.

3.3. Integration and Testing:

- Integrate the QA Service with the Auth Service by ensuring that the QA Service can securely access the document database using the access token obtained from the Auth Service.
- Perform comprehensive testing to verify that both services work seamlessly together. Test various scenarios, including authentication failures, to handle errors gracefully.

3.4. Evaluation and Optimization:

- Design an evaluation framework to assess the performance of your two-staged question answering system
- Compare and evaluate the system performance for different approaches: MapReduce, Refining, Vector store.
- Continuously optimize and fine-tune both the QA Service and the Auth Service. Monitor their performance and address any issues or bottlenecks that arise during testing and usage.

3. 6. User Interface (Optional):

- To add novelty to the solution, we are planning to develop a Web UI based on Python flask using state of the art technologies such as Large Language Models (PaLM v2/GPT 3.5), Vector Stores (Vertex AI), Langchain and Flask framework with 2 interfaces and deploy into serverless functions (if possible):
    - User Query/Chat Interface
    - Data Source Configuration Interface

3.7. Project Inference:

- Provide key findings and project outcomes involving summary, practical implications, interpretations, and contributions.

4. Conclusion:

To conclude, we have initiated the project by laying the groundwork for the Question Answering (QA) and Authentication (Auth) services. This has set the stage for effective document retrieval and user interaction. In the next phase, we will explore LLM models, prepare datasets, integrate, and test the system, optimize it, and develop a user-friendly web UI. The project will culminate in comprehensive outcomes and findings.