CMU Advanced NLP Assignment 2: End-to-end NLP System Building

Anonymous ACL submission

Abstract

Retrieval-Augmented Generation (RAG) combines the strengths of document retrieval systems and generative language models to enhance the accuracy and context-awareness of question-answering tasks. In this project, we developed a comprehensive RAG system specifically designed to answer factual queries related to Carnegie Mellon University (CMU) and Pittsburgh, including their history, culture, trivia, and events. Our methodology involved systematic data collection from publicly available resources such as Wikipedia, municipal websites, and event calendars. We employed advanced embedding models and conducted thorough experiments to optimize retrieval parameters. Evaluating precision, recall, F1 scores, and semantic similarity across various question categories, we demonstrated the effectiveness of leveraging affordable, best-in-class open-source models for building efficient, scalable RAG systems.

1 Introduction

Generative language models like GPT, Gemini, and LLama have shown impressive capabilities in natural language understanding and generation tasks. However, these models often fall short in providing precise and reliable responses for domain-specific questions due to their limitations in training data coverage and lack of dynamic information updates. Retrieval-Augmented Generation (RAG) addresses these challenges by combining retrieval mechanisms with generative models to fetch external, domain-relevant documents, thus improving answer quality and accuracy.

For this course project, we constructed an end-to-end RAG system tailored to answer questions about CMU and Pittsburgh. The project involved collecting, processing, and structuring raw data from a variety of sources including official city pages, university histories, and local event information. We utilized robust open-source tools and advanced embedding models available through platforms like HuggingFace, emphasizing affordability and accessibility. Through extensive experimentation with embedding models and retrieval settings, we investigated trade-offs between precision, recall, and contextual accuracy. Our findings provide actionable insights into creating efficient, reliable, and scalable RAG systems using widely accessible resources.

2 Data Collection

The data collection process for our Retrieval-Augmented Generation (RAG) system involved a systematic methodology for gathering, parsing, and structuring text data from websites and PDF documents. This strategy aimed to balance breadth of coverage, relevance of content, and maintainable scalability.

2.1 Web Crawling

To collect website data, we implemented a Breadth-First Search (BFS) web crawler in Python (Olston et al., 2010). BFS was selected because it is able to discover and process links level by level, reducing the risk of overly deep traversal and allowing fine-tuned controls on how many subpages to include. We employed the following key strategies:

1. **Depth and Page Limits:** For structured sources like Wikipedia, we restricted BFS to a depth and page limit of 1. This approach enabled precise curation: relevant Wikipedia pages were chosen manually, and the crawler fetched only those pages. By contrast, on extensive sites like the official Pittsburgh municipal website, we allowed up to 300 pages at a depth of 2, ensuring comprehensive coverage of subpages without overwhelming system resources.

2. **Link Filtering:** We integrated regex-based filtering to discard URLs containing patterns related to user accounts (`login`, `signin`, `profile`), site settings (`lang=`, `translation`, `sessionid=`), or other extraneous and non-informative pages (`help`, `faq`, `contact`). This targeted filtering minimized irrelevant data, improving the overall quality of crawled content.

3. **Robust Error Handling:** Network timeouts, broken links, and non-HTML responses (e.g., images or archives) were gracefully handled, ensuring that the crawler continued running even when encountering issues like HTTP errors.

4. **PDF Discovery:** If a page linked to PDF files, our crawler automatically detected and downloaded them for subsequent text extraction. This feature consolidated data collection under a single pipeline and ensured PDF resources were captured alongside relevant webpages.

5. **Text Extraction and Cleaning:** After retrieving HTML, we used BeautifulSoup to strip non-informative elements (scripts, headers, footers). The result was a cleaned textual corpus representing each crawled page.

This BFS-based crawling and filtering pipeline offered the flexibility to target specific sections of large sites and systematically broaden exploration on smaller or more specialized sources, thereby balancing data volume with relevance.

2.2 PDF Document Processing

In addition to PDF discovery during web crawling, we also manually added PDF files to a dedicated folder (data_sources/files/). PDFs were processed using the `pdfplumber` library, which was chosen for its reliability and accuracy in extracting text from diverse document structures. The extracted text underwent further cleaning, including:

- Removing extraneous whitespace, special characters, and formatting artifacts.
- Normalizing inconsistent casing and spacing to streamline downstream processing.

These steps produced standardized textual data that was then integrated into the same pipeline as the HTML content, ensuring uniform handling of all documents regardless of source.

2.3 Events Scraping

Downtown Pittsburgh, campus, and Pittsburgh city paper events are easy to scrape using the above methods. For CMU events calendar, for each day, it constructs an API request with relevant query parameters and headers, retrieves event data in JSON format, and processes timestamps into Eastern Daylight Time (EDT). For Pittsburgh events calendar, for each month, we use the package `selenium` to launch a Chrome WebDriver, waits for elements to load, and interacts with "Show More" buttons or expandable sections using JavaScript clicks. After fully expanding the content, it extracts the visible text and appends it to output.

2.4 Automated Q&A Generation

We used LLama 3.1 8B parameters(Touvron et al., 2023) model to generate question-answer (Q&A) pairs from the extracted text. To accommodate the model limits, each document was divided into chunks of approximately 3000 words. We crafted a Q&A-oriented prompt that instructed the model to produce concise, factual pairs directly traceable to the chunk content.

Although chunking was necessary for managing context lengths, it introduced potential limitations: if important information (e.g., the name of an institution) appeared only in a preceding chunk, the model's subsequent chunks might refer to that institution generically (e.g., *"the university"*), leading to incomplete Q&A pairs. Future improvements may involve consolidating repeating context in each chunk or adopting more advanced text-segmentation strategies.

2.5 IAA Evaluation

We randomly choose a subset of 39 problems from our dataset and have two members annotating the answers of the questions. Then we evaluate the similarity between two answers (metrics are described in 5.2) and results are presented in Table 2, which indicates a general agreement on two versions of answers.

Precision	Recall	F1 Score	FNR	SS
0.650410	0.736007	0.666205	0.263993	0.788818

Table 1: Performance Metrics by Category

3 Retrieval-Augmented Generation (RAG) System

Retrieval-Augmented Generation (RAG) combines a document retrieval mechanism with a generative language model to ground responses in relevant source texts (Lewis et al., 2020). As shown in Figure 1, the RAG workflow has two primary stages:

- **Retrieval:** A query is embedded and searched against a corpus of indexed documents to find top matching passages.
- **Generation:** The retrieved passages are provided as additional context to the language model, which uses them to generate an informed response.

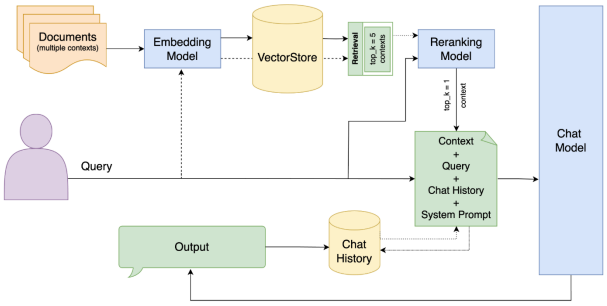

Figure 1: Overview of Retrieval-Augmented Generation (RAG) architecture.

4 RAG System Implementation

Our system uses a modular three-tier design: a React-based frontend, a Node.js middleware layer, and a Python backend responsible for retrieval and embedding. This configuration fosters clear separation of concerns, scalability, and easier testing.

4.1 Architecture Overview

Figure 1 illustrates the main components and data flow in our Retrieval-Augmented Generation pipeline:

1. **Query Input:** A user submits a query to the system, through a chat-style interface.
2. **Embedding:** The query is converted into a numerical vector using an embedding model (e.g., `sentence-t5-base` (Ni et al., 2021), `gte-Qwen1.5-7B-instruct` (Li et al., 2023)). Documents in the corpus have already been pre-embedded and stored in a vector database.
3. **Vector Store and Retrieval:** FAISS stores all document embeddings. When a query arrives, its embedding is compared to the stored vectors, and the most relevant documents or passages are retrieved, for this, we use the cosine similarity measure.
4. **Reranking (Optional):** An additional reranking model can reorder the retrieved candidates for higher accuracy, although our current setup may skip this step if not required.
5. **Context Assembly:** The user query, top-ranked contexts, any chat history, and the system prompt are combined into a single prompt.
6. **Generation:** A language model (such as GPT-4o) processes the compiled prompt to produce a context-aware response, which is then returned to the user.
7. **Chat History:** The newly generated response is appended to the conversation history to maintain continuity in subsequent interactions.

This architecture ensures that information is drawn from relevant documents, reducing unsupported answers and improving factual accuracy. The modular design also allows for swapping out various components (e.g., different embedding or reranking models) without altering the rest of the pipeline.

4.2 Frontend Implementation (React)

The React frontend provides an interactive user interface, managing query input, real-time visualization of results, and chat-style interactions. Key design elements include:

- **Axios Integration:** To communicate with the middleware via structured JSON payloads.
- **Responsive UI Features:** Clear distinction between user messages and AI-generated responses, loading indicators, and adjustable font and spacing for better accessibility.

4.3 Middleware Implementation (Node.js)

We employed an Express.js server to handle RESTful endpoints, bridging user requests from the React frontend to the Python backend. This middleware:

- Abstracts the logic for querying multiple language models via a common interface.
- Maintains conversation state, enabling context-aware responses over multiple turns.
- Provides robust error handling and logging, ensuring a stable user experience.

4.4 Document Retrieval and Vector Search

For document retrieval, we used FastAPI, chosen for its asynchronous event loop and straightforward handling of concurrent requests. FAISS (Facebook AI Similarity Search) (Douze et al., 2024) was integrated for indexing and searching embeddings, enabling efficient sub-linear retrieval times. We selected FAISS primarily because:

- It caches and indexes document embeddings in-memory, eliminating the need to re-embed each query.
- It supports rapid similarity searches, important for timely responses.
- It integrates well with Python, aligning with existing data processing workflows.

Embeddings were generated via `sentence-t5-base` (Ni et al., 2021) and `gte-Qwen1.5-7B-instruct` (Li et al., 2023).

4.5 Deployment and Configuration

Deployment was handled through shell scripts (`deploy.sh`, `install.sh`, and `run.sh`) that facilitate:

- Dependency installation across Python and Node.js environments.
- Environment variable configuration for sensitive information (e.g., API keys).
- Automated startup of all servers in the correct sequence.

Although containerization (e.g., Docker) could streamline future scaling, our current setup focuses on simplicity and quick prototyping.

5 Experiments

5.1 Experiment Setup

We experimented with different model variations.

5.1.1 Embedding Model

We experimented with two different embedding models that yields different embedding dimensions. We hypothesized that more sophisticated embedding model would result in better answers. The models are:

- `sentence-t5-base` (Ni et al., 2021): The model is from sentence-transformer framework. It is a light-weighted model designed for sentence-to-sentence tasks. It is optimized for semantic similarity.
- `gte-Qwen1.5-7B-instruct` (Li et al., 2023): The model is from Alibaba-NLP. It is an enhanced embedding model with bidirectional attention mechanisms, instruction tuning, and training on a vast, multilingual text corpus spanning diverse domains and scenarios. It is expected to generalize well to retrieval tasks.

5.1.2 Top-k Retrieval

We experimented with different number of top matches being retrieved from faiss index. We hypothesize that this will lead to a trade-off between precision and recall: less matches being retrieved will generate more concise and exact information for chat model, but risk of missing information; more matches being retrieved will generate more comprehensive information for chat model, but risk of noise.

5.2 Evaluation Metrics

For evaluation of the generation answers, we employed following evaluation metrics.

5.2.1 Word Occurrences

Given the ground truth answer and the model's answer, we first remove stop words and record the occurrence of each remaining words for calculation of confusion matrices. We define:

- True Positive: the word appeared in both ground truth and model's answer.
- False Positive: the word appeared in model's answer but not in ground truth.
- False Negative: the word appeared in ground truth but not in model's answer.
- True Negative: not considered for evaluation.

Given the TP, FP, FN, we calculate Precision, Recall, and F1 score.

5.2.2 Semantic Similarity

We also evaluated the semantic similarity between the model's output answer and the ground truth. We derived the embedding of two answer using `all-MiniLM-L6-v2` (Reimers and Gurevych, 2019) from sentence-transformer and then calculates the cosine similarity between the two embeddings.

5.3 Results

5.3.1 Embedding Models and Top-k

The result is presented in Figure 2.

5.3.2 Question Category

We divide the questions into 4 categories defined as:

- **Augment_Better**: The question might be answerable with your general knowledge, but reliably verifying or enhancing the answer would be better with additional documents (for example: "What is the name of the annual pickle festival held in Pittsburgh?").
- **Augment_Required**: The question almost certainly requires external, specialized, or updated information to get an accurate answer because it isn't typically part of broad public knowledge (for example: "When was the Pittsburgh Soul Food Festival established?").
- **LLM_Ready**: The question can likely be answered accurately from general knowledge in ab LLM's training (for example: "When was Carnegie Mellon University founded?").
- **Temporal_Sensitive**: The question's answer depends on real-time or future information (for example: "Who is performing at [X event] at [X time]?"), so the response may change based on the date or the event schedule.

The results are presented below, where "FNR" denotes the False Negative Rate and "SS" represents Semantic Similarity.

Category	Precision	Recall	F1 Score	FNR	SS
Augment_Better	0.340502	0.705291	0.422443	0.294709	0.576602
Augment_Required	0.241225	0.560715	0.311222	0.439285	0.458609
LLM_Ready	0.189542	0.799094	0.283966	0.200906	0.449567
Temporal_Sensitive	0.304004	0.734018	0.406808	0.265982	0.532379

Table 2: Performance Metrics by Category

5.4 Analysis

5.4.1 Embedding Model

Recall that the model from Alibaba is more sophisticated at complex language tasks and output a higher dimension embedding, and the model from sentence-transformer(ST) is more light-weighted and output a lower dimension embedding, the results in Fig 2, though similar in general, reflects their difference in capability. The Alibaba model is capable of retrieving relevant content at top-1 setting, but the precision drops afterwards. On the other hand the ST model is relatively incapable of precisely embedding information which lead to incapability in precisely retrieving related contents, so there are improvements in all metrics with an increase of top-k selection. Overall, for semantic similarity, Alibaba's model constantly excel the ST model. This aligned with our understanding of the model and indicates that the capability of embedding model influences the generation of answer by influencing the quality of provided context to chat model.

5.4.2 Top-k Selection

By applying top-1 selection, we are only forwarding the most related content to the chat model. This effectively improved the precision of answer given that there is less noise in the context (for Alibaba's model only; for ST model, we think the low precision with top-1 selection indicates its inability to precisely embedding information). Overall, the model benefits from changing to top-3 selection which increase tolerance on problems in embedding and retrieval and offer more related context for the chat model. We also observed a precision-recall trade-off between different top-k selection.

5.4.3 Question Category

Table 2 indicates that the model is better at Augment_Better and Temporal_Sensitive than the rest two categories. For Temporal_Sensitive problems, the model benefits from retrieval which provides critical real-time updates, enabling more precise answers. For Augment_Better problems, the model also benefits from retrieval which adds more relevant information for answering the question. The relatively low performance in Augment_Required problems indicates possibility in the model's lack of capability in embedding and retrieval or the dataset for retrieval is missing the required information. For LLM_Ready problems, as we expected,

performed the worst given it's not benefited from the retrieval.

References

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. *arXiv preprint arXiv:2401.08281*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith B Hall, Daniel Cer, and Yinfei Yang. 2021. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. *arXiv preprint arXiv:2108.08877*.

Christopher Olston, Marc Najork, and 1 others. 2010. Web crawling. *Foundations and Trends® in Information Retrieval*, 4(3):175–246.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *Preprint*, arXiv:1908.10084.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

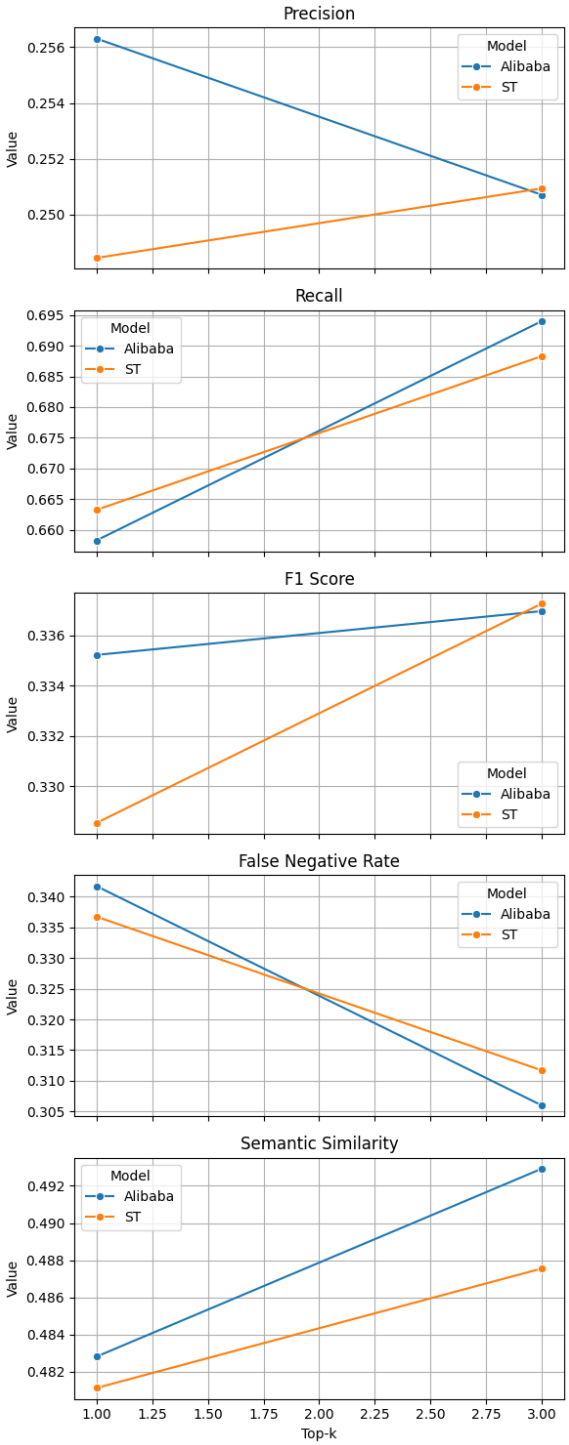

Figure 2: Comparison between Embedding Models and Top-k