11711 ANLP HW 2: E2E CMU RAG

Cyprien Riboud-Seydoux, Chris Su, Yan Cai
Carnegie Mellon University
Pittsburgh, PA, USA
`{cribouds, chrissu, yancai}@cmu.edu`

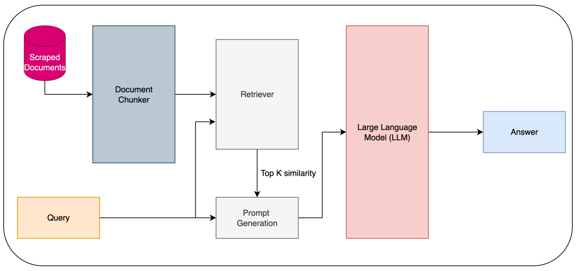

Figure 1: End to End Pipeline

1 Introduction

This report documents our 11-711 Advanced NLP HW 2 (L3, 2025) project submission, for an end-to-end retrieval augmented generation system (Gao et al., 2023). Our team is: Cyprien Riboud-Seydoux, Yan Cai, and Chris Su. Our end to end pipeline consists of the following: a document chunker, a chunk retriever / embedder, and finally the prompt generation that gets fed to the large language model. In the appendix, you may find our prompt.

2 Data

2.1 Data Collection

Data was collected using a web scraping algorithm. In particular, we collected several links that we decided were relevant for our retrieval augmented generation system. Then, these links were split into two subsets. The first subset, we call a "one jump" links, and the second, "zero jump" links.

The scraper collects links to scrape with the following methodology: add all zero jump links to the links to scrape. Also add all links in "one jump" links to links to scrape. Finally, for each link in "one jump" links, take one jump (i.e., go to each sublink) from that link, and also scrape that text.

We distinguished between zero jump and one jump links by using heuristics. For example, if we believed that taking one jump from the provided link would be self-contained within the webpage, then we added that link to the one jump link section. Otherwise, we added it to the zero jump link subset. For instance, the Pittsburgh wikipedia webpage (Wikipedia, 2025b), because of its very broad outreach to other wikipedia webpages, was added to zero jump subset. However, the Carnegie Mellon wikipedia webpage (Wikipedia, 2025a) was added to the one jump subset.

We used this heuristic for all links provided within the original "links to scrape" set by 11-711 HW 2, "CMU Advanced NLP Assignment 2: End-to-end NLP System Building" Github (L3, 2025).

Finally, for each link that was collected, we scraped only the text on the webpage. The limitations of this are that information in images likely are not scraped. No other preprocessing / filtering / processing was performed after scraping this data from the internet.

In the appendix, we have included the main scraping algorithm pseudocode [B] as well as the zero jump and one jump links [C]. This program was multi-threaded in Python, and utilized the BeautifulSoup4 package (Richardson, 2025).

2.2 Data Annotation

After scraping the webpages with the above algorithm, we were able to collect text files from about 732 webpages. We then randomly sampled 150 text files, split amongst three of our members (to be 50 webpages each), and created one question and answer pair for each webpage. This formed the test set for validating our retrieval augmented generation (RAG) system. To validate our annotations, we conducted the IAA protocol. In particular, we randomly sampled 30% of our test set questions, along with the associated chunk of text that the question was created from, and re-answered the question with the information in the chunk. We used both the exact match metric and the F1 score metric across this random sample. These metrics came out to 65.12% and 74.74%. We manually annotated our data, using no annotation interface.

2.3 Training Data

No additional data was used / procured to train or tune our model.

3 Model Details

The underlying embedding model we used was **Linq-AI-Research/Linq-Embed-Mistral** (Junseong Kim, 2024). The model was released in 2024, and we used the model to generate query and document embeddings for our dense retriever. The model has about 7 billion parameters (Junseong Kim, 2024).

For our large language model, we used Meta's Llama 3.1 8 billion parameter model, which was released in July, 2024. The baseline used only Meta's Llama 3.1-8b model (Grattafiori et al., 2024), allowing us to quantify the improvements made by retrieval-augmented generation. We selected this model due to its strong balance between efficiency and performance, making it suitable for our resource constraints and desired iteration times whilst still being competitive on open-domain question answering tasks.

All implementation was done using the sentence-transformer (Reimers and Gurevych, 2019), python library, along with models provided by Hugging Face (hug).

Table 1: RAG System Configuration

Parameter	Value
Temperature	0.4
Top-k documents	5
Window Length	300
Overlap Length	100
Retriever embedding size	2048
Batch size	8
Sample	True
Token Truncate	50
Embedder	Linq-Embed-Mistral
LLM	Llama 3.1 8b

For our retriever implementations, we tried and experimented with sparse, dense, and hybrid retrieval approaches. The sparse retriever used the open-source implementation of BM25 (Lù, 2024), which relies on lexical overlap. This method is effective for proper nouns and numeric values. The dense retriever used FAISS (Johnson et al., 2019) for similarity search over embedding vectors, complementing the sparse method by capturing semantic similarity. This enabled a secondary level of retrieval where the query and document do not share exact forms or phrasing structures. Our hybrid retriever combined both of these approaches by using score normalization and weighted averaging, mitigating the greatest weaknesses of each individual method.

4 Results

We evaluated our models on the annotated test set using the standard metrics of Exact Match (EM) and F1. Our baseline model was the sparse retriever, BM25. This sparse retriever was compared to the dense (semantic-aware) counterpart as well as the hybrid retriever. We tried these methods as this would make the comparison's the most fair. In the perspective of the model, the LLM generation is retriever agnostic, in the sense that all chunks of text that can be retrieved will always be the same chunk. The method of retrieval does not actually change what the chunk looks like. That is, if both retriever A and retriever B retrieve chunk c, then c will be the same piece of text across both prompts. Hence, we can have a comparable and testable scenario between all three retrieval methods and run statistical tests about which retrieval method seems to be the strongest strategy. We provide the closed-book results for reference.

Retrieval Method	EM	F1
Dense (FAISS + Linq-Embed-Mistral)	4.00	15.19
Sparse (BM25)	1.33	12.70
Hybrid (Dense + Sparse)	**4.67**	13.10
Baseline (Closed-book)	4.00	**20.42**

Table 2: Evaluation results for different retrieval strategies. EM = Exact Match, F1 = token-level F1 score.

Dense retrieval: Using FAISS and the Linq-Embed-Mistral model, our system achieved **4.0 EM** and **15.19 F1**.

Sparse retrieval: Using BM25, our system achieved **1.33 EM** and **12.7 F1**.

Hybrid retrieval: Combining dense and sparse results yielded **4.67 EM** and **13.1 F1**.

Baseline (Closed-book): The closed-book model achieved **4.0 EM** and **20.42 F1**.

4.1 Statistical Significance

We applied the statistical significance testing procedure described in the report by performing paired

Table 3: Per-group EM and F1 (%) by retrieval method.

	HYBRID		SPARSE		DENSE		BASELINE	
Group	EM	F1	EM	F1	EM	F1	EM	F1
Misc.	5.56	17.97	0.00	15.66	0.00	10.45	0.00	4.33
In *	0.00	10.11	0.00	16.90	0.00	6.81	0.00	13.47
What *	6.00	17.15	3.85	13.30	2.67	18.82	3.56	22.11
What is *	**12.00**	**20.55**	0.00	8.35	**8.00**	**22.72**	**8.00**	27.47
When *	0.00	0.00	0.00	5.56	0.00	2.63	0.00	4.29
Where *	0.00	6.35	0.00	16.67	0.00	9.23	0.00	25.47
Which *	0.00	16.93	0.00	10.71	7.89	16.30	7.89	**31.05**
Who *	0.00	10.87	**16.67**	**21.67**	0.00	14.56	0.00	18.23

testing on EM and F1. The difference between sparse and dense retrieval was statistically significant, confirming that dense retrieval outperforms sparse retrieval for our test data set. However, the improvement of hybrid retrieval over dense retrieval was not statistically significant, showing that whilst hybrid retrieval yielded higher average scores, these performance improvements may not generalize beyond this test data set.

5 Analysis

The dense retriever performed moderately well in identifying semantically related passages, reflected in its higher F1 but low EM, suggesting partial correctness without exact lexical overlap. In contrast, the sparse BM25 retriever struggled with paraphrased queries due to its reliance on exact token matches, though it successfully captured some direct overlaps.

The hybrid approach slightly improved EM by combining the strengths of both retrieval types—BM25's precision on entity-heavy queries and dense retrieval's broader semantic coverage—though the overall gain in F1 remained limited.

5.1 Closed-Book vs. RAG

Finally, the closed-book baseline, which relies purely on the model's parametric knowledge, achieved the highest (raw) F1 score, indicating that the model could often generate contextually relevant answers even without external retrieval. There are two hypotheses for this.

First, our web scraping method likely did not retrieve enough (or disproportionately few) content concerning questions outside of the model's training distribution (such as questions about recent or upcoming events). Therefore, the ability for the RAG system to shine was not pronounced in this case.

The second is that F1 score / EM score may not be able to distinguish between good answers and answers that *sound* good.

5.2 Question Group Analysis

To more closely investigate model failure modes, we looked into performance across question groups. In particular, we sorted the test question set in alphabetical order, and then merged the questions by group or type of question. The final types that we created are: **Misc., In *, What *, What is *, When *, Where *, Which *, Who ***. Then, amongst each group, we computed the F1 and EM scores, for each retrieval method that we implemented. We found that the hybrid retrieval method is particularly good across the board, but excels in answering "What is ..." questions. Sparse retrieval methods are good at answering "Who ..." questions.

5.2.1 Example: "Who ..." Question (Sparse)

```
Question: Who is UPitt's 18th chancellor?

Answer (Sparse): "Patrick D. Gallagher was named
      the 18th chancellor of the university and
     assumed the office on August 1, 2014. [ 32
     ]"

Answer (Dense): "Joan Gabel"

Answer (Hybrid): "Joan Gabel\n"

Answer (Baseline): "Mark Nordenberg, who served
     from 1995 to 2016, was UPitt's 18th
     chancellor.\n\n"
```

One interesting artifact is that since many sources / document chunks come from wikipedia, the model often hallucinates giving "sources" such as "[32]", even though these sources may not even exist in the original material. The correct answer was Patrick D. Gallagher.

5.2.2 Dense and Hybrid Similar Behavior

Dense performs similarly to hybrid retrieval in the sense that it performs well across the board, but also excels at "What is..." questions. The baseline (with non RAG system) is good at answering "Which ..." questions, but generally outperforms most question categories than RAG systems except for in "When ..." questions, which sparse retrievers perform best in.

5.2.3 Example Model Outputs

"Not Found" Response (Dense Retriever)

```
Question 7: Who invented the first micro electro
     mechanical system?

Not found.
```

The model returns "Not Found" if no evidence was found. Often times, however, we found that even though the document was inside the prompt, it would hallucinate that it could not find the answer. We believe one hypothesis is that encouraging the model to answer in short, concise responses, may make the model weight responses like "Not Found", which is short, to be higher probability, over retrieving the answer which is in the document.

5.2.4 Best Performing Retriever Across Question Groups

In this section, we briefly go over the best performing retriever across the different question groups aforementioned. The full table for per-group EM and F1 can be found in Table 3. In particular, what we also find is that Sparse retriever performs well in scenarios that all Hybrid and Dense retrievers fail in. The runner up is the Dense model, and finally Hybrid. We can see that this analysis might reveal a little more information than previous analyses that simply took the mean across F1 and EM scores for all question types.

Table 4: Per-group EM and F1 (%) winners excluding BASELINE.

Group	Winner Method	EM	F1
Misc.	Hybrid	5.56	17.97
In *	Sparse	0.00	16.90
What *	Dense	2.67	18.82
What is *	Dense	8.00	22.72
When *	Sparse	0.00	5.56
Where *	Sparse	0.00	16.67
Which *	Dense	7.89	16.30
Who *	Sparse	16.67	21.67

5.2.5 Best Performing Retriever vs. Baseline

Lastly, we look into how these retrievers compare against the baseline. Once again, we can see that sparse retrievers perform well in many categories vs. the baseline with no RAG system.

Table 5: Winners per question group based on F1 (%). Bold indicates the best method.

Group	Winner (F1)
Misc.	Hybrid (17.97)
In *	Sparse (16.90)
What *	Baseline (22.11)
What is *	Baseline (27.47)
When *	Sparse (5.56)
Where *	Baseline (25.47)
Which *	Baseline (31.05)
Who *	Sparse (21.67)

6 Generative AI Use

We used Generative AI to write a web scraper that took in a series of links and would output raw text files from HTML/pdf. We also used Generative AI to assist in writing some of the scripts for parsing the data we collected.

7 Bibliography

References

Hugging face. http://huggingface.co/. Accessed: 2025-10-10.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *Preprint*, arXiv:2312.10997. Accessed: 2025-10-10.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Jihoon Kwon Sangmo Gu Yejin Kim Minkyung Cho Jy-yong Sohn Chanyeol Choi Junseong Kim, Seolhwa Lee. 2024. Linq-embed-mistral:elevating text retrieval with improved gpt data through task-specific control and quality refinement. Linq AI Research Blog.

CMU L3. 2025. Cmu advanced nlp assignment 2: End-to-end nlp system building. https://github.com/cmu-l3/anlp-fall2025-hw2. Accessed: 2025-10-10.

Xing Han Lù. 2024. Bm25s: Orders of magnitude faster lexical search via eager sparse scoring. *Preprint*, arXiv:2407.03618. Accessed: 2025-10-10.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Leonard Richardson. 2025. Beautifulsoup4. https://pypi.org/project/beautifulsoup4/. Version 4.14.2. Accessed: 2025-10-10.

Wikipedia. 2025a. Carnegie mellon university. Accessed: 2025-10-10.

Wikipedia. 2025b. Pittsburgh. Accessed: 2025-10-10.

A Prompts

A.1 Non-RAG Prompt (Baseline)

```
def prompt(query):
    return f"""
You are a factual question-answering assistant.

Help me answer this question: {query}

Instructions:
- Answer concisely in one sentence or less.
- Do not explain or justify your answer.

Answer:
```

A.2 RAG Prompt

```
def prompt(query, chunks, k):
    chunk_str = "\n\n".join([f"[CHUNK {i+1}]\n{chunk}" \
    for i, chunk in enumerate(chunks)])
    return f"""
You are a factual question-answering assistant.

You will receive a single user query and
{k} retrieved document chunks that may contain
the answer.

Read them carefully and extract the most
relevant information to answer the query
directly.

If the answer is not found in the documents,
reply with: "Not found."

==================== DOCUMENTS ====================
{chunk_str}
==================== END DOCUMENTS ================
Query: {query}
Instructions:
- Answer concisely in one sentence or less.
- Do not explain or justify your answer.
- Do not repeat or reference the documents.
- If uncertain or no evidence is present, say "Not found."

Answer:
```

B Web Scraper

```
##############################################
# One Jump Scraper Pseudocode
##############################################

FUNCTION scrape_webpage(link, folder_path):
    CREATE_DIRECTORY_IF_NOT_EXISTS(folder_path)

    response = HTTP_GET(link, headers=USER_AGENT_HEADERS, \
                        timeout=30)
    soup = PARSE_HTML(response.text)

    REMOVE_TAGS(soup, ["script", "style"])
    text = EXTRACT_TEXT(soup)
    text = CLEAN_WHITESPACE(text)

    parsed_url = PARSE_URL(link)
    domain = REMOVE_PREFIX(parsed_url.domain, "www.")
    path = REPLACE(parsed_url.path, "/", "_")
    base_name = CONCAT(domain, "_", path)
    base_name = SANITIZE_FILENAME(base_name)

    file_path = folder_path + "/" + base_name + ".txt"
    counter = 1
    WHILE FILE_EXISTS(file_path):
        file_path = folder_path + "/" + base_name + "_" + \
                    STRING(counter) + ".txt"
        counter += 1

    WRITE_FILE(file_path, text)
    RETURN (True, "")
END FUNCTION


############################################################

FUNCTION one_jump_collector(link):
    response = HTTP_GET(link, headers=USER_AGENT_HEADERS, \
                        timeout=30)
    soup = PARSE_HTML(response.text)

    links = []
    FOR anchor IN FIND_ALL_TAGS(soup, "a", attribute="href"):
        href = anchor["href"]
        absolute_url = MAKE_ABSOLUTE_URL(link, href)
        IF absolute_url STARTS_WITH "http://" OR "https://":
            links.APPEND(absolute_url)

    RETURN links
END FUNCTION


############################################################

FUNCTION scrape(zero_jump_links, one_jump_links,
                scrape_info_path,
                folder_path,
                max_workers = 10):
    start_time = CURRENT_TIME()

    all_links = SET(zero_jump_links + one_jump_links)
    collected_links = []

    WITH ThreadPoolExecutor(max_workers) AS executor:
        futures = {executor.submit(one_jump_collector, link):
                    link FOR link IN one_jump_links}
        FOR future IN AS_COMPLETED(futures):
            collected_links.EXTEND(future.result())

    all_links.UPDATE(collected_links)
    sorted_links = SORT(LIST(all_links))

    CREATE_DIRECTORY_IF_NOT_EXISTS(PARENT(scrape_info_path))
    WRITE_LINES(scrape_info_path, sorted_links)

    successful_scrapes = []
    failed_scrapes = []

    WITH ThreadPoolExecutor(max_workers) AS executor:
        futures = {executor.submit(scrape_webpage, link, \
                    folder_path): link FOR link IN \
                    sorted_links}
        FOR future IN AS_COMPLETED(futures):
            (success, error_msg) = future.result()
            IF success:
                successful_scrapes.APPEND(link)
            ELSE:
                failed_scrapes.APPEND((link, error_msg))

    log_path = PARENT(scrape_info_path) + "/scrape_results.txt"
    WRITE_LOG(log_path,
        total_links = LENGTH(sorted_links),
        successful = successful_scrapes,
        failed = failed_scrapes
    )

    elapsed_time = CURRENT_TIME() - start_time
    RETURN (successful_scrapes, failed_scrapes, elapsed_time)
END FUNCTION
```