A Retrieval-Augmented QA System for Pittsburgh and CMU

Chenglin Zhang
Carnegie Mellon University
chengliz@andrew.cmu.edu

Qingyang Liu
Carnegie Mellon University
qliu3@andrew.cmu.edu

Haojun Liu
Carnegie Mellon University
haojunli@andrew.cmu.edu

Abstract

In this project, we develop a retrieval-augmented generation (RAG) system tailored for factual-checking and question-answering about Pittsburgh and Carnegie Mellon University. Recent advancements in large language models (LLMs) have significantly improved performance across a range of natural language tasks, yet challenges such as outdated knowledge and hallucination remain. To address these limitations, our approach combines LLM generation with highly relevant retrieved documents, ensuring more accurate and contextually informed answers. We conducted extensive data collection and cleaning, created a high-quality question-answer annotation dataset, and carefully fine-tuned each component of the RAG pipeline, including model selection, document sharding strategies, and prompt engineering. Our system demonstrates notable improvements over traditional non-RAG baselines, and a qualitative analysis highlights key challenges that affect the effectiveness of RAG-based solutions.

1 Overview

The project workflow includes 4 major components: data acquisition, QA-pair annotation, RAG pipeline implementation, and evaluation analysis as shown in Figure 1.

For the data acquisition part, we decided to manually collect relevant URLs to construct our knowledge base. For each web page, we decided to collect one layer of sub-links. The crawled data for the manually collected URLs will be used for both the RAG document base and QA pairs generation, while the crawled data for sub-links will be used for the RAG document base only.

For the question-answering with the RAG pipeline part, we decided to adopt the method of using an LLM in a few-shot learning setting. This will require us to choose a base LLM, document embedding and retrieval methods as well as prompt engineering.

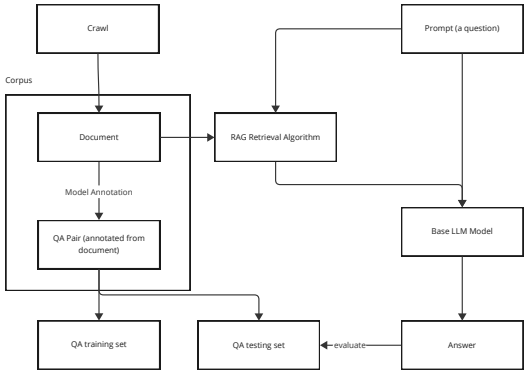

Figure 1: The pipeline overview

For the QA-pair annotation part, we decided to automate this process by using an LLM and prompt engineering to automatically extract question-answer pairs from each document. The generated QA-pairs will be manually evaluated with the IAA pipeline and will be used to evaluate the performance of the QA with the RAG pipeline.

For the evaluation part, we decided to adopt the 3 metrics mentioned in the SQuAD paper (Rajpurkar et al., 2016), which are Exact Match, Answer Recall, and F1. The questions of generated QA pairs will be fed into the QA with RAG pipeline. The generated results will then be compared with the answers from the QA pairs to calculate the three metrics.

2 Data creation

2.1 Raw Data Crawling

We split the topics of the relevant web resources into 5 main categories: General Info & History, Events after Oct. 27, Culture, Music, and Sports. In the beginning, we decided to manually collect 40 relevant URLs for each category, 20 for the city of Pittsburgh, and the other 20 for Carnegie Mellon

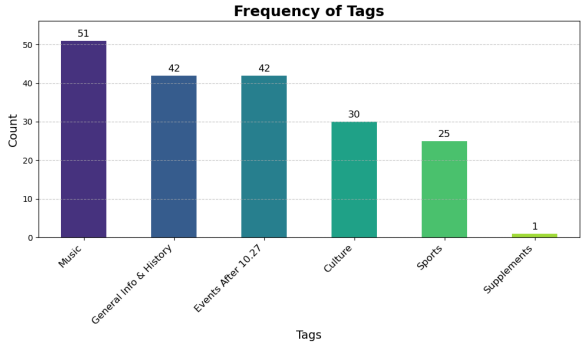

Figure 2: Category frequency. Note that each URL may have multiple categories so the sum will exceed 173.

University (if there is enough information). We collected 173 URLs in total in the tabular format which includes the category of the URL, the type (web page or PDF), a short description of the URL, and the collector of the URL (one of the team members). The detailed table can be found through this link. An overview of the category distribution for these links can be found in Figure 2.

The Python package BeautifulSoup4 was initially used for automatic data collection. We originally intended to extract the most relevant information using HTML tags of **<h1>, <h2>, <h3>, <h4>, <h5>, <p>**.

A manual check was conducted for each crawled text file for the 173 URLs. Every web page (excluding the PDF links, which will be mentioned later) was successfully crawled. However, after the initial crawling, a lot of relevant information in HTML tags like ** ** was not crawled because of the variance of the web pages' structures. Thus, we changed the crawling strategies to include all the text data on each web page.

Some websites (especially for those events websites that often have a calendar component, of the 173 URLs there are 42 of them) require enabling JavaScript to correctly load all the contents, which static website crawling methods won't work. Thus, a dynamic web crawler is needed to correctly extract information for these websites. In our case, Selenium was used for extracting the data for the 42 events websites. A manual check was also conducted for the crawled information and if there was still information missing, we manually replaced the texts.

To filter out irrelevant information on each web page, the Boilerplate Removal technique (Leonhardt et al., 2020) was initially adopted. However, after manual checking, it even filtered out important relevant information. Thus, eventually, we decided to keep all the crawled text data.

For PDF web links, the Python package pypdf was used to extract the text data in each PDF file. There are 13 PDF files in total in the 173 URLs.

During the crawling of these 173 URLs, we collected the hyperlinks that appeared on each web page. There were 18005 unique sub-links initially. After some URL cleaning (for example, *https://en.wikipedia.org/wiki/Pittsburgh* and *https://en.wikipedia.org/wiki/Pittsburgh#History* are considered the same URL as they direct to the same web page), there were 13889 valid unique URLs remaining. A static crawler using BeautifulSoup4 was used to crawl these web pages. This created a document set of about 200MB in size.

2.2 Data Annotation

2.2.1 Data Choice for Annotation

As previously mentioned, we do not perform any model training, so the annotations are solely for testing and evaluation. Our annotation dataset, consisting of question-answer pairs, is generated from the cleaned data extracted from 173 URLs that were manually searched and audited. We select these sources as the primary resources for QA annotations because their content has been manually reviewed and is highly relevant to the task. By limiting the scope of annotation resources to a carefully chosen set of documents, we aim to minimize the risk of generating irrelevant or trivial questions that would not effectively assess the performance of our RAG system.

To determine the size of the annotation dataset, we conducted a simple grid search to identify the optimal number of QA pairs generated from a document chunk by using an LLM. We observed that the model typically generates fewer than 10 highly relevant QA pairs per chunk. In total, we collected **3938** QA pairs from 760 chunks of documents. Comparing this with standard QA benchmark datasets as shown in Table 1, we justify this target size of ~4,000 QA pairs for evaluation.

Dataset	Training Size	Evaluation Size
SQuAD 1.1	~10,000	~1,300
SQuAD 2.0	~12,000	~5,700
Natural Questions (NQ)	~8,800	~3,600
Ours	–	~4,000

Table 1: QA Dataset Statistics

2.2.2 Annotation Interface

We prompt an LLM - Llama-3.1-8B-Instruct - to generate QA pairs from each chunk of documents. Due to the limited computing resources, we could not fully exploit the maximum context length of the model to handle extremely long documents. Therefore, we performed a manual document sharding and created document chunks with a maximum word count of 3K. The sharding is sentence-based using the package NLTK. A new sentence will be appended to the current chunk if the chunk's word count will not exceed 3K. For each chunk, it also has 300 words overlap with the previous and next chunk respectively. We insert the document chunk into a prompt template in a few-shot learning setting, Below are a few key prompting instructions we use to enhance QA generation quality:

- To avoid duplicate QA pairs:

  Based on this document, generate 5-10 factual question-and-answer pairs ... If there are fewer pairs available, only provide the number you can find.

- To avoid questions being too vague:

  The questions should make sense independently of the document. For example, the question "Q: What is the fee of the event?" is not acceptable, because words like "the event" is too vague without context of a document.

We have noticed the duplication of questions generated from different document sources, yet the LLMs may give slightly different answers each time. We perform de-duplication of QA pairs based on questions and maintain all unique answers as a list of reference answers, which align with the format of the hidden test set on which our system will finally be evaluated. after performing the post-processing, we shortened the size of the annotation dataset by 323 rows, and 64 questions have more than one reference answer.

To evaluate the performance of the RAG QA pipeline on different kinds of questions, we decided to split the questions into two groups based on whether the question is asking about a future event. Such splitting was also implemented by prompting Llama-3.1-8B-Instruct. **856** questions are labeled as questions asking for future events out of the 3938 questions.

2.2.3 Annotation Quality Evaluation

We adopt the suggested assessment of inter-annotator agreement (IAA) to evaluate the quality of the model-generated QA pairs. We randomly sampled 50 QA pairs from our 3938 set, and Chenglin Zhang and Haojun Liu independently rated the QA pairs' question relevance and answer correctness.

The range for the question relevance is $[0, 1, 2]$, where 0 suggests the question is irrelevant to Pittsburgh or CMU, 1 means the question is related but should add a bit more clarifications (like the question *What is the total projected expenditures of the Special Summer Food Service program in 2024?*), and 2 means that the question is highly related and with enough clarifications that can be answered directly. The range for the answer correctness is $[0, 1]$ (incorrect/correct), and is only available for questions whose relevance is either 1 or 2.

Chenglin Zhang flagged 35 out of 50 questions as relevant, while Haojun Liu flagged 40 out of 50. The Cohen's Kappa for question relevance is 0.334. The Cohen's Kappa for answer correctness is 0.793. This suggests that there was fair agreement on the question's relevance (question's quality w.r.t. clarification/enough context), while for answer correctness, there was almost perfect agreement.

3 Model details

We employ the Langchain framework to implement our RAG models. We adopt the **Llama3.1-8b-Instruct** as our backbone LLM for answer generation and experiment with different types of custom text splitters, embedding models, document retrieval methods, and additional RAG-advanced strategies including **Reranking** and **Hypothetical Document Embeddings** (Gao et al., 2022).

3.1 Baselines

3.1.1 Close-book Model

As the most straightforward baseline, we include one close-book model using the same backbone LLM as we used for our RAG pipeline. By comparing it with the no-RAG pipeline, we hope to evaluate the effectiveness of our RAG-based QA systems in answering questions that the state-of-the-art LLMs may not be capable of answering with their pre-training knowledge. Below is the overview of the baseline and the 3 different strategies.

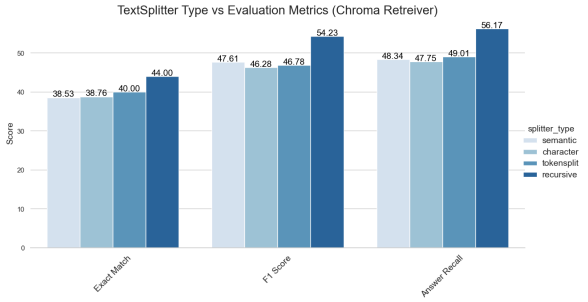

Figure 3: Splitter comparison

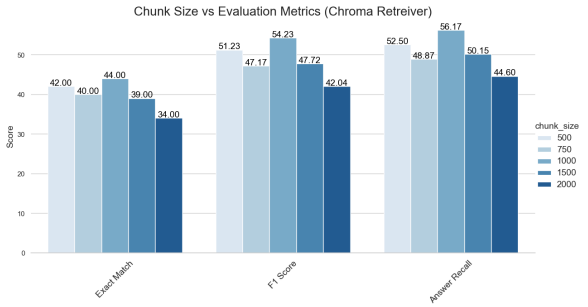

Figure 4: Chunk Size vs Evaluation Metrics (Chroma Retriever)

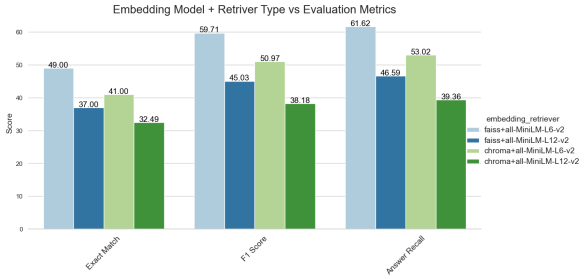

Figure 5: Embedding Model + Retriver Type vs Evaluation Metrics

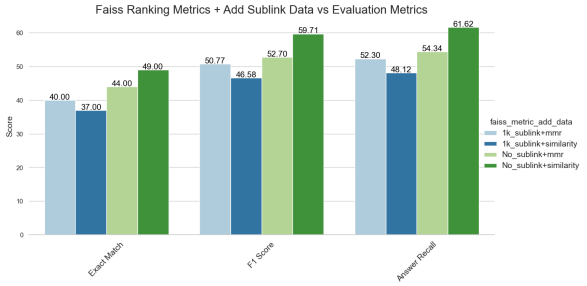

Figure 6: Faiss Ranking Metrics + Add Sublink Data vs Evaluation Metrics

3.1.2 Default RAG Model

We develop a simple Rag pipeline in Langchain with standard components necessary for a RAG system including a text splitter for document sharding, an embedding model to build document embeddings, and a retriever to retrieve the most relevant document chunks for a given query. To get a decent starting point, multiple sets of experiments were conducted to select the optimal configurations. For each hyper-parameter group, controlled variable experiments were run on the same 100-sample QA set from the 3938 annotated QA set. The results were evaluated by the 3 metrics from the SQuAD paper as mentioned earlier.

Splitter Type To determine the best text splitter, experiments were conducted on four different splitters with the same chunk size of 1000 and overlap number of 200 (note that the meaning of the number will be different for each splitter): character split, semantic split, token split, and recursive split. Results shown in Figure 3 indicate the recursive split achieves the highest score for all of the three metrics.

Chunk Size To determine the optimal chunk size for the recursive splitter, experiments were conducted on five different chunk size groups with the overlap being the same as 200. Results in Figure 4 indicate that chunk size 1000 achieves the highest score out of the 5 groups.

Embedding Model + Retriever The options of the embedding models are all MiniLM-L6-v2 and all MiniLM-L12-v2 from SentenceTransformers. The options of the retriever are Chroma and FAISS. Both retrievers' ranking algorithm is set to default. Results in Figure 4 indicate that FAISS consistently outperforms Chroma and all-MiniLM-L6-v2 outperforms all-MiniLM-L12-v2.

Ranking Algorithm + Corpus Size As we fixed our choice of using the FAISS and all-MiniLM-L6-v2, the options for the ranking algorithms are the default built-in similarity search algorithm and Maximal Marginal Relevance (MMR) (Carbonell and Goldstein, 1998). As previously mentioned, the sub-link corpus will be used to expand our document set for handling a wider variety of questions. Thus, a further group of tuning was conducted on the effect of the ranking metrics interacting with the corpus size of retrieved documents. Remember that so far we are using the primary 173 URLs for retrieval, while we also have a bulk of documents scraped from sublinks of 173 parent web pages.

As shown in Figure 6, we observed that there is a robust advantage of retrieval without documents from sublink webpages (the green bars) regardless of the type of ranking metrics. This counter-

intuitive observation makes sense here since all QA data for evaluation are automatically generated from the 173 primary URLs. Therefore, an augmentation of sublink documents introduces noises to the retrieval process, which likely causes the score to drop.

As the corpse grows, the maximal marginal relevance (MMR) metric performs better than the cosine similarity one, which can be attributed to MMR's tendency to retrieve a more diverse document list (Carbonell and Goldstein, 1998). The benefits of the integration of sublink documents + MMR will become obvious for the unseen question set from the teaching group when manual evaluation of the generated answers that our RAG system produced in our submission.

Prmopt Template It is also worth mentioning here that we carefully engineer our prompting template to make sure the LLM makes full use of the retrieved contexts to answer the question. Some key points of our prompt engineering efforts are listed here:

- To avoid "lost-in-the-middle": we adopt the sequence of **Instruction**, **Few-shot Example**, **Context**, and **Question**, and we concatenate the top-k contexts in ascending order to the ranking score so that the most relevant context chunk will be placed closest to the question being asked.

- To reduce hallucination, we use instructions: *If the provided context does not contain the answer, leverage your pretraining knowledge to provide the correct answer. If you truly do not know, just say "I don't know."*

- To generate concise answers, we use instructions *Do not use complete sentences for answers. Provide only the word or phrase that directly answers the question. For example, given the question "When was Carnegie Mellon University founded?", you should only answer "1900".*

3.2 Advanced Variation

Now that we have figured out the best recipe for the default RAG baseline. The configurations for the default baseline are

- RecursiveTextSplitter with chunking size = 1000, chunking overlap = 200

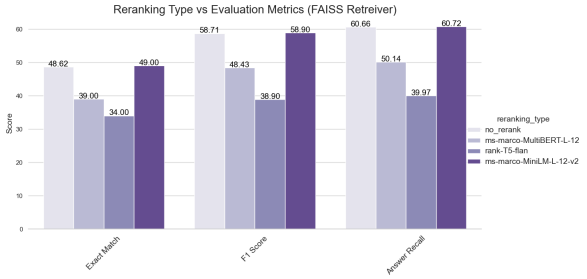

Figure 7: Reranking Type vs Evaluation Metrics (FAISS Retriever)

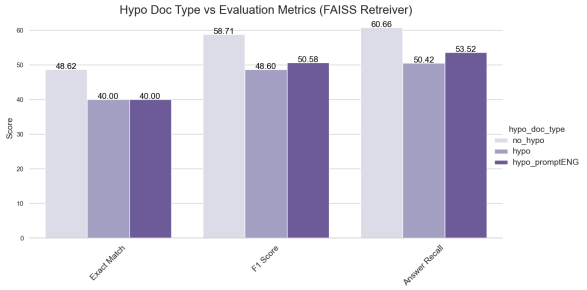

Figure 8: Hypo Doc Type vs Evaluation Metrics (FAISS Retriever)

- Embedding: all-MiniLM-L6-v2 (dim=384)

- FAISS retriever with similarity, top-k = 3

Built upon this, we propose two advanced methods that can potentially boost the RAG performance.

3.2.1 Reranking

The rerank strategy in search retrieval for question answering (QA) involves an initial retrieval step where relevant documents or chunks are selected based on their similarity to the query, often using vector-based methods like cosine similarity. Once this initial set of top documents is retrieved, the rerank strategy applies a more sophisticated model, typically a large language model (LLM), to re-evaluate and reorder the retrieved documents based on their relevance to the query. This process helps refine the results by pushing the most contextually appropriate answers to the top, often considering deeper semantic relationships and understanding beyond just surface-level keyword matches.

In our experiments, we adopt the FlashRank (Damodaran, 2023) method incorporated in LangChain to perform the reranking. We experiment with a list of reranker models supported by FlashRank, including ms-marco-MultiBERT-L-12 (the default reranker model), ms-marco-MiniLM-L-12-v2 (cross-encoder reranker), and rank-T5-flan

(non cross-encoder reranker). We do give a shot to LLMs as the reranker such as the zephyr-7b-v1, but due to the limit of cumpute resource, a light-weighted reranker is observed to be the best solution in our project scope. As shown in 7, Using ms-marco-MiniLM-L-12-v2 yields a slightly better performance than the no-reranking baseline, while all other rerankers perform worse. One hypothesis is that MiniLM-L-12-v2 belongs to the same model family of the embedding model MiniLM-L-6-v2, which should have more similiar embedding spaces, leading to better alignment between the retrieval and reranking steps. This suggests that the synergy between the embedding model and the reranker plays a crucial role in optimizing retrieval quality.

3.2.2 Hypothetical Document Embeddings

Hypothetical Document Embeddings (HyDE) (Gao et al., 2022) is a retrieval method that uses "fake" (hypothetical) documents to improve the answers generated by LLM. This method leverages an LLM to generate a hypothetical answer given a query and uses the vector embedding of this hypothetical document to retrieve the real documents in the vectorstore. While this artificially created document might contain inaccuracies, it encapsulates patterns and nuances that resonate with similar documents in a reliable knowledge base, and therefore likely to improve the accuracy of retrieval.

In our experiments, we still use the Llama-3.1-8B-Instruct to generate hypothetical documents given the question. We also conduct fine-grained prompt engineering, where we force the LLM to output the original question if it does not know relevant information about the input question. The results of no-HyDE baseline, HyDE method with a naive prompt template, and HyDE with fine-grained prompt engineering are shown in Figure 8. Using HyDE leads to an obvious performance drop in all three metrics, which is likely due to the fact that the LLM has minimal knowledge of the majority of questions we annotate, which leads to bad quality of hypothetical documents, and a misleading retrieval search (examples can be seen in Table 2, where hallucination still occurred for the close-book model which might hinder the retrieval process). The modification in the prompting template only provides trivial improvement from its naive counterpart.

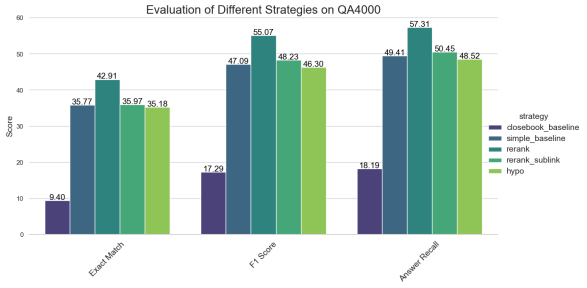

Figure 9: Evaluation of different strategies on the entire 3938 QA set

4 Results

The evaluation of different strategies on our entire QA dataset of 3938 entries yields notable differences in performance across three key metrics: Exact Match, F1 Score, and Answer Recall (Figure 9). 3 experiments plus the closebook baseline have been run and the demonstrated values were the average scores for each evaluation metrics.

Across all metrics, the rerank without any sublink corpus strategy demonstrates significantly better performance on our entire QA set by a margin of approximately 7-8 percentage points. The strategies of reranking with 10K sublink corpus and the default RAG baseline have no significant difference in performance for the exact match score, while for F1 and Answer Recall, the sublink strategy is about 1 percentage point better. The hypothetical answer strategy, on the other hand, performed slightly worse. However, the RAG pipelines demonstrated a substantial improvement compared with the closed-book LLM in answering the questions. Though for our test set, the strategy of reranking without any sublink corpus yields the best results, from manual evaluation of the generated results for the unseen test set, the strategy of MMR + 10K sublink might generate better results. This is still awaiting verification of our submissions.

5 Analysis

To gain a deeper understanding of the intrinsic differences among the three strategies, we conducted a more fine-grained experiment. Intuitively, a model incorporating both a retriever and a ranker is exposed to more relevant contexts that can support answer generation compared to a closed-book model. Furthermore, the inclusion of a reranker further enhances the quality of the selected contexts. To elucidate the differences in context selection, we divided our test cases into two categories: questions

Table 2: Comparison of our rank, rerank, and closed-book LLM's answers for a selected group of questions from our self-built test set. All answers from the rerank model are correct.

Question	Closed-book LLM	Rank RAG pipeline	Rerank RAG pipeline
What time is the Billy the Kid's Steel Town All-Stars performing at The R Bar?	December 24	Sundays, 7-10 p.m.	Sundays, 7-10 p.m.
When is the New York Jets at Pittsburgh Steelers game?	20-Oct-24	10/20	10/20
What time is the Milwaukee Panthers at Duquesne Dukes Basketball game on Nov 19?	I don't know	I don't know	7:00 PM
Which team is playing at Petersen Events Center on February 02?	Pittsburgh Panthers	North Carolina Tar Heels	Virginia Cavaliers

Table 3: Comparison of Future and Non-Future Events using Rank and Rerank Methods For the 3938 QA set

Method	Event Type	Exact Match	F1 Score	Answer Recall
Closed-book Method	Non-Future Event	11.38	20.36	21.51
	Future Event	2.80	7.21	7.40
Default Method	Non-Future Event	40.66	51.68	54.13
	Future Event	19.08	31.36	33.25
Rerank Method	Non-Future Event	48.20	60.05	62.32
	Future Event	25.53	38.76	40.88

about future events and those concerning general knowledge, as mentioned in section 2.2.2. Since future-event questions typically require more extensive contextual information compared to general knowledge questions, we were able to effectively visualize the context selection differences through a controlled comparison.

The analysis across the three methods (Baseline, Rank, and Rerank) revealed significant performance differences between future event and non-future event questions. For all evaluation metrics (Exact Match, F1 Score, and Answer Recall), non-future event questions consistently achieved higher scores. This disparity highlights the challenge of retrieving sufficient context for future event questions. The Baseline Method showed a substantial performance gap, while the Rank Method improved scores for both question types but still exhibited a notable gap. The Rerank Method further narrowed the gap, especially for future event questions, with improvements in F1 Score and Answer Recall. Notably, the performance ratio between non-future and future events decreased from Baseline to Rerank, indicating the effectiveness of the reranking step in reducing disparity. Notably, the ratio between non-future and future event performance decreased as the strategy evolved from Baseline to Rerank, indicating that the reranking step effectively mitigates the performance disparity.

Viewing from Table 2, the close-book LLM can hardly answer hard questions involving future information. The rank RAG pipeline can answer some of them, while the rerank RAG pipeline achieves the best performance.

In conclusion, the retrieve-and-augment approach outperforms the closed-book approach, particularly for non-future event questions, as evidenced by the improvement across all metrics. However, a performance gap persists for future event questions, suggesting that future work should focus on enhancing retrieval methods for more contextually complex queries, such as those concerning future events.

References

all MiniLM-L12-v2. https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2.

all MiniLM-L6-v2. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.

BeautifulSoup4. https://beautiful-soup-4.readthedocs.io/en/latest/.

Jaime Carbonell and Jade Goldstein. 1998. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336.

Chroma. https://python.langchain.com/docs/integrations/vectorstores/chroma/.

Prithiviraj Damodaran. 2023. FlashRank, Lightest and Fastest 2nd Stage Reranker for search pipelines.

FAISS. https://github.com/facebookresearch/faiss.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2022. Precise zero-shot dense retrieval without relevance labels. *arXiv preprint arXiv:2212.10496*.

Langchain. https://python.langchain.com/docs/introduction/.

Jurek Leonhardt, Avishek Anand, and Megha Khosla. 2020. Boilerplate removal using a neural sequence labeling model. In *Companion Proceedings of the Web Conference 2020*, pages 226–229.

Llama-3.1-8B-Instruct. https://www.nltk.org/.

NLTK. https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct.

pypdf. https://pypdf.readthedocs.io/en/stable/.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv e-prints*, pages arXiv–1606.

Selenium. https://selenium-python.readthedocs.io/.

SentenceTransformers. https://www.sbert.net/.