Assigment-2 Report

Vashisth Tiwari
vashistt@andrew.cmu.edu

Amanda Li
xal@andrew.cmu.edu

Emily Guo
epguo@andrew.cmu.edu

1 Introduction

We created an effective RAG system that answers questions pertaining to Carnegie Mellon and the Language Technology Institute. Our pipeline consists of data curation and processing, document retrieval, model selection, and answer generation. We provide a thorough analysis of 3 different models we experimented with: FlanT5 (large and xlarge), Llama (zero shot and one shot), and Mistral. Further details on our data processing iterations are also described in this report.

2 Data Creation

At the front of this pipeline, we gathered the data we assumed we needed. We collected a comprehensive list of documents based on the scope of the assignment, specifically those that closely related to CMU and LTI. After a brief review of the released test set, we realize that this may be a false assumption in some cases. For instance, there is a question about whether or not we need a CMU ID to access the Fitness center, which is information that we did not scrape. However, we included all the information listed in the assignment writeup.

1. Webpages: LTI faculty, LTI programs, Spring Carnival, Commencement, 25 Great Things, SCS History, CMU History, Buggy History, CMU Tartans Facts, Mascot Facts, and Kiltie Band

2. PDFs: CMU Academic Calendars, LTI Program Handbooks, More CMU Facts

3. Authors and papers: we grabbed information about main LTI faculty. This included their papers, their author IDs, their citation counts, and so on.

Note: there were two options for collecting the CMU Academic Calendars - HTML or PDF. After parsing the HTML format, we realized that it ended up in a huge chunk of text that we would then have to reformat manually. The PDF parser produced a nicer tabulated version with newlines. (We ended up converting this to natural language anyway but that was a separate decision described later.)

The implementation specifics are described in the following subsection.

2.1 Data Extraction

Given the list of URLs, we proceeded each type of data separately using specific tools, described below. This was pretty straightforward, except for some caveats for faculty duplicates.

2.1.1 Web Scraping

Associated file: `process_html.py`

We used urllib to grab HTML from a website and Beautiful Soup to process HTML, grab plain text, and remove HTML tags (Richardson, 2007; url). This was not perfect, since it inserted some newlines in random places. In addition, the opposite problem occurred. For example, for the Carnival schedule webpage, Beautiful Soup parsing produced a large block of unformatted text. More information on data cleaning is described in the following section.

2.1.2 Faculty Papers + Metadata

Associated Files:
- `semantic_helper.py`
- `semantic_scholar_new.ipynb`
- `semantic_scholar_downloader.py`

Faculty articles were collected using the Semantic Scholar API (Kinney et al., 2023). To do this, we manually went through the Semantic Scholar website to get the author IDs of the faculty members in the LTI Faculty List. Initially, this process was automated; however, we saw that the author IDs were not always correct. Note that multiple faculty members have two Semantic Scholar IDs; the information was recovered from all relevant IDs for a faculty member. To download the articles, we use the batch download functionality of the API that allows us to obtain all the articles that are `openAccess ==True`.

2.2 Data Cleaning & Preparation

2.2.1 PDFs

Associated files:
- `process_pdfs.py`
- `clean_pdfs.py`
- `format_schedule.ipynb`

The two main types of PDFs we incorporated as documents were technical papers from LTI faculty and school-wide academic calendars and schedules; these sources required drastically different data cleaning methods detailed below. For all PDFs, we used the `PdfReader` library to parse the PDFs into `txt` files (Polshcha). Other PDFs included LTI program

handbooks, the CMU factsheet, and academic calendars which were more easily parsed by `PdfReader` compared to papers/schedules.

- **Academic Papers**
  Due to encoding constraints in the LangChain `DirectoryLoader` function, we chose to use `utf-8` encoding and decoding on the PDFs which resulted in poor parsing of mathematical equations and symbols (Chase, 2022). We reasoned that given the purpose of this model as a question-answering system geared towards LTI and CMU, our system would be unlikely to receive in-depth technical questions that rely on equations, proofs, etc. and as such any non-alphanumeric and non-punctuation characters were filtered out.

  The raw text files of the papers required extensive formatting and cleaning as the double-column format of most academic papers created many unnecessary line breaks, hyphenations, and newlines. Most of these excess characters were removed by combining hyphenated words that were interrupted by newlines and removing lines in the file that were below 20 characters; these short lines were generally the result of poor character parsing and no longer contained any useful information.

  Another challenge was the lack of a semantic layer in PDF files; this makes parsing any kind of table extremely difficult, and thus almost all information contained in paper tables was lost during parsing. In conclusion, the notoriously difficult task of parsing PDF files (particularly those with complex formatting like academic papers) resulted in a non-insignificant amount of information lost in data cleaning, which would be reflected in our performance metrics later on.

  An additional processing step we implemented prior to embedding all documents was to prepend the title of the paper to each of the paper's chunks post-splitting. We reasoned that the majority of each academic paper would have no information in each chunk tying back to a specific paper title or author, so we manually added the title of each chunk's source paper to itself after decreasing the chunk size to account for the increase in tokens. This proved beneficial for the correctness of contextual question answers that asked about datasets or methods about specific papers.

- **Schedule of Classes**
  Our document corpus contained the schedule of classes of 4 semesters (Fall 2023, Spring 2024, Summer 1/All 2024, Summer 2 2024). These documents were semi-structured in that they used spaces to enforce a table structure rather than true PDF tables that are represented using symbols and glyphs. As a result, the schedules were more easily parsed into `txt` files.

  A large hurdle we encountered with the schedule of classes content was the format in which to present the information. We experimented with expressing the course information as plain text (unchanged from the original parsing) and as natural language by converting each course's information into question-answer pairs (eg. *"What time does the course 11711 Advanced Natural Language Processing begin? 12:30PM"*). We experimented with different embeddings after it became apparent that our models did not retrieve information from the semi-structure of the plain text schedules; this is likely because the structure from the spaces was destroyed by the document chunking process.

  We expected the natural language format to be much more successful, however, using embeddings from the natural language schedules did not lead to a drastic increase in performance, which did not align with our expectations since the format of the questions in this natural language version strongly resembled the language in test sets (both our own and the released test set). We hypothesized that this was due to two main reasons: 1) We noticed that some course information had been dropped in the process of converting from semi-structured to natural language and 2) Not every question contained text about the specific semester the information was pertaining to, which meant that some chunks of the schedule natural language likely did not contain any semester information at all and negatively impacted the retriever's ability to fetch relevant chunks.

2.2.2 HTML Webpages

There were two main data cleans we did for webpages: reformatting the tables and the Carnival schedule webpage.

- **Tables**

  Tables were the main issue on the LTI program website. The `CSV` format was not understandable. Thus, we manually postprocessed it to annotate the LTI curriculum in a list format.

- **Carnival Schedule**

  Similar to the CMU Academic Calendars, parsing this webpage produced an unformatted large block of text. We manually formatted this page to divide events and descriptions that would make more sense to feed into a model later on.

2.2.3 Paper & Author Metadata (JSONs)

In our sample runs on `Flan-T5`, we saw that `JSONs` were particularly hard for the model to work with.

1. **Problem:** The two main types of issue we saw with the Jsons were the model not being able to associate the content with the file name. For example, even simple questions such as *What is the authorID of Graham Neubig?* were not answered by Llama or Flan-T5. The model did not answer any of the first-author questions because it did not have a notion of first-author. Even after giving an in-shot example, we were unable to consistently get the right answers.

2. **Solutions Considered:**

[I] **Appending Important Pointers to Each Chunk:** We experimented with adding the title of the paper and the LTI faculty name after splitting to embed (more details of embedding in 2.3

[II] **Convering JSONs to Question Answer Format** (Code in `semantic_scholar_json_extraction.ipynb`)
The second method considered was to change the JSON data, which is in dictionary format, to generate QA pairs manually in a specific format. A prespecified list of questions was created in which keys were used in the questions and values were stored as the answers.
Two kinds of question were generated: *author-related data* and *paper-related data*. The generated questions can be found on our Github repo. In the sample run over the questions on JSONs, we saw a significant improvement across all metrics, and hence they were used for the final embedding.

2.3 Embedding Generation

The framework we used was `"thenlper/gte-base"`. gte-embeddings were trainined large-scale corpus of relevance text pairs, covering a wide range of domains and scenarios. Given our hardware restrictions during most of this assignment we went with gte which for us was a good trade-off between the size, speed, and performance. Being able to run the embedding locally in a reasonable amount of time while not sacrificing performance completely was the criterion we use. The embedding were chosen based on the MTEB Leaderboard (Muennighoff et al., 2022) Initially, `gte-small` was used but was discarded in favor of `gte-base` and `gte-large` . Between the two, we did not see much improvement with large and hence `gte-base` was chosen.

2.4 Storing Vectorized Data

To better transport and modulate our document embeddings that we used for both retrieval, we utilized the FAISS (Facebook AI Similarity Search) Vector Store within LangChain VectorStores to save and load the files locally.

2.5 Question Generation

We generated a diverse set of questions that spanned all document types - 70 for questions on webpages (of which 16 were based on tabular information), 41 PDF-based questions (of which 7 were about papers, 11 were about schedules of classes), and 1796 paper metadata and author questions. We curated about 3-10 unique questions for every webpage and PDF, choosing our questions to retrieve data from a variety of presentation formats. Number of questions varied based on the amount of content per document. We took care to have Yes/No questions, data/time specific questions (since there are multiple combinations of MM/DD/YYYY or just YYYY, etc), tabular info questions, questions that required list-formatted answers, and even trivial questions that probably didn't need document retrieval.

Additionally, we automated question generation for author and paper metadata documents. For each field given in the documents, we asked a question where the answer appeared directly in the collected JSON. Both the question and answer were given in complete sentences that a model would be able to understand later on. We also experimented with **Automatic Question-Answer Generation** with LLama 7B Instruct version for papers. However, these questions were not included in the test set due to lack of quality and the time it took us to generate them.

We used all questions from the webpages and PDFs as part of our test set. We also included a random subset of automated questions for the metadata. Finally, we had 191 questions in our test set. This was sufficient to perform significance testing for model comparison. This is likely due to our diverse set of questions not only across the various document types but also our question formats and data reasoning the model would need to perform. Note that we did not have training data as we did not finetune our models for answer generation.

2.6 Inter-Annotator Analysis (IAA)

We used the exact match of the metrics, the F1 score, and the recall to evaluate the agreement between the notators on our manually generated questions. Answers to questions between two annotators were preprocessed to be capitalization and punctuation agnostic, compared one-to-one, and scores averaged to produce our final IAA metrics. This was repeated for questions on documents of different modalities.

IAA metrics for exact match, F1 score and recall on web page documents are 0.4, 0.634, and 0.652, respectively. The same metrics are 0.267, 0.642, and 0.793 for PDF documents and 0.733, 0.8, and 0.833 for JSON documents. The significant difference between the sets of performance metrics on JSON document questions versus the other two modalities suggests a difference in both the question format and document readability. Upon inspecting the JSON questions, we found that they were much more straightforward and direct than the questions generated for webpages and PDFs, which were not as easily answerable by finding exact string matches between the query and the documents.

The interannotator performance metrics aligned

with our expectations prior to conducting the analysis. The questions written for webpages and PDFs were more open-ended, and any ambiguity in these questions was reflected in our answers as annotators occasionally had drastically different interpretations of a question. For example, the question *"What is a Tartan?"* was answered as *"twilled woolen fabric with a plaid design"* by one annotator and *"Carnegie Mellon athletic team member"* by another, both of which are fairly valid without context. Another example of inner-annotator disagreement was the question *"How do I demonstrate English proficiency for a F-1 or J-1 visa?"* The gold answer was *"TOEFL, IELTS, or Duolingo test scores"* but an annotator answered *"Through an official copy of an English proficiency score report and a minimum TOEFL score of 100."* Both responses are factually correct but have little overlap or exact similarity.

From this analysis, we hypothesized that the various RAG models in our experimentation would achieve better performance on questions generated from the JSON documents; the inherent key-pair structure of JSON files likely makes them more suitable for retrieval than raw text. Similarly, as demonstrated in the examples in the previous paragraph, questions generated from less structured documents are more ambiguous, and thus we hypothesized that these types of questions will incur comparably poor performance.

Document	Metric		
	EM	F1	Recall
Webpages	0.4	0.634	0.652
PDFs	0.267	0.642	0.793
JSONs	0.733	0.8	0.833
All	0.547	0.717	0.733

Table 1: Inner-Annotator Analysis Metrics

3 Model Selection and Comparison

3.1 Retriever and Reranker

Our retrieval and reranker pipeline is a two staged process which first uses a non-interactive approach (cosine similarity) and then a late-interaction approach (ColBERTv2). ColBERTv2 was accessed using the implementation in ragatouille (Clavié).

For retrieval, we used `DistanceStrategy.COSINE` (cosine similarity) for the distance strategy; which returns the cosine similarity between the query and the vectorized documents.

Although this was effective for simpler queries (webpages), for most other questions, we saw that simply having the retriever was not enough. Therefore, ColBERT-v2 was introduced a more efficient version of "**ColBERT**" which stands for Contextualized Late Interaction over BERT, to rerank our retrieved documents (Khattab and Zaharia, 2020; Santhanam et al., 2022). More details on how this strategy was performed, cases in which we needed it, and how we optimized the number of documents retrieved and reranked are provided in Section 4.

3.2 Models Considered for Reader LLMs

For the project, we experimented with a variety of models. The models considered were

1. `Flan-T5-large` and `Flan-T5-xlarge`.

   For most of our tests, `Falan-T5-large` served as the baseline model. This was because the model was small enough to run on our local machines and gave very reasonable results, despite being small in size.

2. `Llama.cpp` version of quantized instruction tune Llama2 (`llama-2-7b-chat.Q4_K_M.gguf`) (Chung et al., 2022; Gerganov, 2023; TheBloke, 2023)

3. `TheBloke/Mistral-7B-Instruct-v0.1 (Q5_K_M)` via Huggingface

4. * `Gemma 2B IT`: This model was only used for initial exploration, but was discarded due to the lack of well-quantized versions, with not good enough performance on our initial albeit small sample test. We also found that it is considerably slower for inference than other models of comparable parameters (Flan-T5 and Llama2).

3.3 Reasoning for choosing these models

The models were chosen on the basis of the following criteria.

- **Architecture:** Flan-T5 models are fine-tuned versions of T5 models, which follow an encoder-decoder architecture. However, Llama2 and Mistral are decoder-only architectures. We wanted to have at least one from each of these architectures. The specific choice of Llama2 and FLAN-T5 was inspired by the paper "Learning to Filter Context for Retrieval-Augmented Generation" (Wang et al., 2023). Mistral was chosen for its performance in OpenLLMLeaderboard.

- Availability of instruction tuned models on Huggingface

- **Disk Size:** All the models considered take less than 5 GB of disk space. The only exception to this rule is FLAN-T5-xl, which is a 3B parameter and occupies $\sim$ 13 GB storage. This was only added as an exception to this requirement due to how well FLAN-T5-large performed, and we wanted to see if xl could do significantly better.

- **Availability of Quantized Versions & Llama CPP Avaialability** Even a full-precision 7B LLama2 and Mistral are upward of 13.5 GB disk space. Given the limitation of `g4dn.xlarge` on AWS, which has a memory of 16 GB, these models were out of our bandwidth.

Therefore, the 8-bit quantized versions which are both ~ 5 GB in memory were used.

Llama CPP is a lightweight and portable implementation of various open-source LLMs that allows faster inference (Gerganov, 2023). Both of these models are available in different quantized versions. Given our requirement of a 5 GB disk size, `K_M.gguf` versions of these models were chosen.

4 Results & Analysis

The results of our investigation in this assignment are summarized in Tables 2, 3. In this section, we also present some of the observations we have made throughout this assignment.

1. Table 2 contains the results regarding EM, F_1, and Recall Across Different Models and Settings

2. Table 3 contains the results regarding the Significance test on the performance of models in different metrics

We compared models that produced statistically significant results (see Table 3). Additionally, our models with document retrieval have statistically significant results from their out-of-box versions. The FlanT5 and LLama (1 shot) models do noticeably better than their no-context counterparts. Our Mistral document retrieval model works significantly better in F1 and recall. Details and examples can be found in the following section.

4.1 Importance of Data Preparation

We conducted many iterations on our data embeddings that we passed in because we realized that different formats would impact answer generation. Below are some of the key ideas that were made.

4.1.1 Metadata to paper

As mentioned before, when we prepared our data and transform them into embeddings, we chunk them into lengths of 512 tokens. This means that it is very easy to lose information for faculty research papers because the information is now separated. To combat this, we added the title of the paper as metadata prepended to each paper chunk.

There was experimentation on what kind of metadata we should prepend to papers. At first, we thought that title, list of all authors, and paper ID. However, because of the token limit of each chunk, we chose to include just title as it was the most informative aspect out of the three.

For instance, one of our test questions was *"What datasets were used in the paper "Enhancing End-to-end Conversational Speech Translation Through Target Language Context Utilization"?"*. When our data was purely chunked papers, the model responded with *"<unk> 'DBLP': 'journals/corr/abs-2309-15686', 'ArXiv': '2309.15686', 'DOI': '10.48550/arXiv.2309.15686', 'CorpusId': 263152687<unk>"*. As we can see, there was irrelevat information and weird formats in this answer. However, once we added our metadata to the paper chunks, the RAG responded with the correct answer: *"Fisher-CallHome Spanish English [33], IWSLT22 Tunisian Arabic-English [34], and BOLT Chinese-English [35]"*.

4.1.2 JSONs and Schedules to Natural Language QA format

We did not run any quantitative tests, but we ran the models on two sets of embeddings (processed into natural language, and without processing). Across the questions we had chosen, we saw that for JSONs the model was able to almost always get the right documents and mostly output the correct answer. Even thought schedule-related questions received results that are lower compared to other categories (see Table 2), we saw a big improvement between the processes and non-processed data.

- The first author questions were only answered by the model after we included a QA pair for each paper in our documents. This makes sense because the model might not have an idea of what the first author means.

- Without QA processing, the model struggled with the authorId and HIndex questions. For example, the JSON metadata includes the authorIDs of each of the authors of a paper. The model will repeatedly output the authorIDs (or other similar numerical values) in place of the correct authorID.

- For the schedule, the greatest improvement was seen in the questions related to the schedule.

4.2 Retrieval and Reranker Optimization

We saw that while the initial settings of 3 and 2 worked fine when had embeddings pertaining to only specific data types, there was a significant drop in performance after we combined all of the FAISS indexes. The clash was most visible in the questions related to JSONs which referred to the title of the paper, and the cosine similarity retriever was grabbing mostly the paper chunks.

We had to increase the number of documents retrieved from 3 to 5, but that was not enough. In the end, we found that retrieving 10 documents and then reranking with colbert, and then choosing the top 3 worked fine for us in the limitation of the context windows of the models.

4.3 Metrics across Different Models

1. **Verbosity and its Effect on the exact match and F_1 Scores**

   LLama and Mistral were quite a bit more verbose in their outputs compared to both FLAN-T5 models despite promoting brevity in the

Category	Model	EM	F1	Recall
Overall	FLAN-T5 Large	0.370	0.523	0.542
	FLAN-T5 XLarge	0.398	0.570	0.593
	LLama 2 7B Chat†	0.094	0.260	0.510
	LLama 2 7B Chat	0.050	0.208	0.483
	Mistral 7B	0.022	0.234	0.561
Webpage	FLAN-T5 Large	0.444	0.640	0.658
	FLAN-T5 XLarge	0.426	0.656	0.708
	LLama 2 7B Chat†	0.074	0.299	0.636
	LLama 2 7B Chat	0.093	0.316	0.694
	Mistral 7B	0.000	0.233	0.697
Tabular Webpage	FLAN-T5 Large	0.062	0.253	0.344
	FLAN-T5 XLarge	0.188	0.425	0.455
	LLama 2 7B Chat†	0.000	0.096	0.442
	LLama 2 7B Chat	0.000	0.144	0.466
	Mistral 7B	0.000	0.139	0.429
Non-Paper PDF	FLAN-T5 Large	0.087	0.250	0.243
	FLAN-T5 XLarge	0.087	0.292	0.327
	LLama 2 7B Chat†	0.000	0.154	0.258
	LLama 2 7B Chat	0.000	0.133	0.264
	Mistral 7B	0.000	0.222	0.427
Paper PDF	FLAN-T5 Large	0.000	0.057	0.159
	FLAN-T5 XLarge	0.000	0.166	0.238
	LLama 2 7B Chat†	0.000	0.039	0.079
	LLama 2 7B Chat	0.000	0.057	0.222
	Mistral 7B	0.000	0.075	0.222
Schedule PDF	FLAN-T5 Large	0.000	0.152	0.114
	FLAN-T5 XLarge	0.000	0.253	0.189
	LLama 2 7B Chat†	0.000	0.126	0.121
	LLama 2 7B Chat	0.000	0.016	0.045
	Mistral 7B	0.000	0.093	0.136
JSON	FLAN-T5 Large	0.571	0.688	0.702
	FLAN-T5 XLarge	0.629	0.720	0.723
	LLama 2 7B Chat†	0.186	0.344	0.614
	LLama 2 7B Chat	0.057	0.210	0.490
	Mistral 7B	0.057	0.299	0.630

Table 2: Model Performance Metrics with Categories († indicates one-shot prompting)
The best performance metrics in each category are highlighted in red for EM, blue for F1, and green for Recall.

prompts. This led to close to 0 EM and very low F_1 scores. This could also be a byproduct of how these models are trained in the chat format, where they are expected to give more verbose answers rather than just key points.

For example, consider the question from our test set: *"Who taught the class 11741 Machine Learning with Graphs in Spring 2024?"*. FLAN-T5 produced the answer *"Yang"* while Llama stated *"The class 11741 Machine Learning with Graphs was taught by Yang in Spring 2024."* Our reference answer only stated the professor name, as suggested by the assignment guidelines. We are unsure if a student were to be using our model in practice, they would prefer the more succinct or more complete answer.

2. **Recall**

   LLama, FLAN-T5 XLarge, and Mistral have a higher recall, which means they can get the right information. Higher recall but lower F_1 is not only because they always get the right answer, sometimes we see these models outputting large chunks that contain a lot of unnecessary information along with the correct one.

4.4 Categorical Performance

Across all categories, FLAN-T5 XLarge almost always outperformed other models for all three. However, as highlighted above, we saw that all the models (Llama, FLAN-T5 XL, and Mistral) were quite close in the recall values. As mentioned in Section 4.3, both FLAN-T5 models were significantly more concise in their answers and this is one of the reasons for their comparably higher exact match scores.

We also see that JSONs are the highest performing category in all models. This shows that our preprocessing of JSONs into natural language has leads to a considerable improvement. Schedule PDFs, despite attempts to convert to natural language, show poor performance. This is due to two things: the formatting of the webpages before processing and some lost data in trying to automate processing. We tried to automatically convert the schedule webpages into QA format; however, we saw due to inconsistency across different sections of the webpages that this

Comparison (Sys1 vs Sys2)	Metric	Win %(1)	Win %(2)	Tie ratio	Better Model
Llama (1 shot) vs FlanT5	EM	1.000	0.000	0.000	Sys1, p = 0
	F1	1.000	0.000	0.000	Sys1, p = 0
	Recall	0.969	0.031	0.000	Sys1, p = 0.031
FlanT5 (xl) vs Mistral	EM	1.000	0.000	0.000	Sys1, p = 0
	F1	1.000	0.000	0.000	Sys1, p = 0
	Recall	0.772	0.228	0.000	Sys1, p = 0.228
Llama (1 shot) vs Mistral	EM	0.956	0.019	0.025	Sys1, p = 0.044
	F1	0.708	0.292	0.000	Sys1, p = 0.292
	Recall	0.106	0.894	0.000	Sys2, p = 0.106
RAG Llama (1 shot) vs closed book	EM	1.000	0.000	0.001	Sys1, p = 0.000
	F1	1.000	0.000	0.000	Sys1, p = 0.000
	Recall	1.000	0.000	0.000	Sys1, p = 0.000
RAG FlanT5 (xl) vs. closed book	EM	1.000	0.000	0.000	Sys1, p = 0.000
	F1	1.000	0.000	0.000	Sys1, p = 0.000
	Recall	1.000	0.000	0.000	Sys1, p = 0.000
RAG Mistral vs closed book	EM	0.863	0.019	0.138	Sys1, p = 0.137
	F1	1.000	0.000	0.000	Sys1, p = 0.000
	Recall	1.000	0.000	0.000	Sys2, p = 0.000

Table 3: Significance Testing of Different Models. The better model contains the better system and the p-value of the corresponding win

could not completely capture the whole page. We hoped that, in combining the structured and the processed data, we will be able to generate a rich enough embedding for question-answer purposes.

The model also struggles with papers despite adding the metadata. We hypothesize that this is due to both the nature of the questions that we were asking (which were highly specific and can need context from different chunks that are not immediately next to the retrieved document) and how the papers are filled with a lot of words that are not very relevant to the query, making it hard for the retriever and LLM.

4.5 Sensitivity of Llama and Mistral to Prompts

In our test set, we experimented with Llama with a one-shot example (for formatting and brevity) and zero-shot. The example in the prompt was

> Again we just want the main key points in the answers. For example, if the question is 'What is the biggest planet in the solar system?'; the answer should be 'Jupiter'

As we would expect, we see a big difference in the EM metric.

With Mistral, we chose the template that performs the best. We do not have quantitative measures that quantify the difference in performance, but we saw that there was a big difference. We noticed the greatest difference in the outputs on the test questions and experimented with prompts that seemed to render results that made the most sense format-wise.

For all models, we experimented a lot with prompting and despite the low temperatures of 0.2 saw significant variations in the output. Thus in our analysis, the gap between `Flan-T5's` and other models could be further reduced if we were to experiment more with better prompting strategies.

4.5.1 RAG vs No-Context Model Performance

It is clear from Table 3 that our RAG models perform significantly better than the out-of-the-box versions. This shows that having a context is crucial for accurate model answers.

For instance, we see that given a CMU-specific question like *"What is the purpose of fairings on a buggy?"*, the closed-book version responds *"to protect the driver"*. From an outsider's perspective, given general knowledge of what a buggy is during training, this is a reasonable answer. However, only from the buggy article that we included can we conclusively determine that the right answer would be *"reduce drag, make the vehicle quieter and just looks cool"*.

We also briefly note that Mistral does the best out-of-the-box of the 3 models we evaluated. This can probably be attributed to its larger training size and the number of parameters. It is also the newest model out of the 3. We also note that, across all categories, the closed-book models performed best on the webpages. It is possible that the models, in particular mistral, saw some of the CMU webpages data in its training.

References

Library: urlib.

Harrison Chase. 2022. LangChain.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi,

Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models.

Benjamin Clavié. [link].

Georgi Gerganov. 2023. llama.cpp: Llm inference in c/c++.

Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert.

Rodney Michael Kinney, Chloe Anastasiades, Russell Authur, Iz Beltagy, Jonathan Bragg, Alexandra Buraczynski, Isabel Cachola, Stefan Candra, Yoganand Chandrasekhar, Arman Cohan, Miles Crawford, Doug Downey, Jason Dunkelberger, Oren Etzioni, Rob Evans, Sergey Feldman, Joseph Gorney, David W. Graham, F.Q. Hu, Regan Huff, Daniel King, Sebastian Kohlmeier, Bailey Kuehl, Michael Langan, Daniel Lin, Haokun Liu, Kyle Lo, Jaron Lochner, Kelsey MacMillan, Tyler Murray, Christopher Newell, Smita R Rao, Shaurya Rohatgi, Paul Sayre, Zejiang Shen, Amanpreet Singh, Luca Soldaini, Shivashankar Subramanian, A. Tanaka, Alex D Wade, Linda M. Wagner, Lucy Lu Wang, Christopher Wilhelm, Caroline Wu, Jiangjiang Yang, Angele Zamarron, Madeleine van Zuylen, and Daniel S. Weld. 2023. The semantic scholar open data platform. *ArXiv*, abs/2301.10140.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*.

Maksym Polshcha. Library: pdfreader.

Leonard Richardson. 2007. Beautiful soup documentation. *April*.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. Colbertv2: Effective and efficient retrieval via lightweight late interaction.

TheBloke. 2023. Thebloke/llama-2-7b-chat-gguf.

Zhiruo Wang, Jun Araki, Zhengbao Jiang, Md Rizwan Parvez, and Graham Neubig. 2023. Learning to filter context for retrieval-augmented generation.