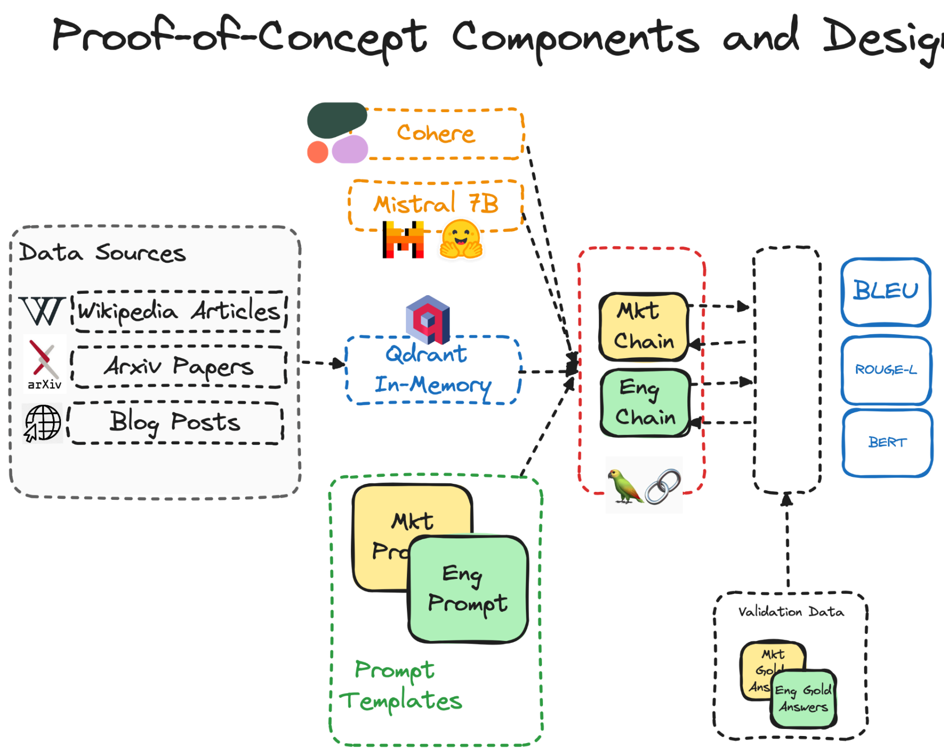
Proof-of-Concept for a Large Language Model (LLM) based Data Retrieval Augmented Generation (RAG) System

13 Apr 2024 **-** Submitted by **Eshwaran Venkat** (eshwaran@ischool.berkeley.edu)
Part of deliverables on A5 for the course MIDS 290 (Generative AI) at
University of California - Berkeley School of Information - Spring 2024 Term
Instructors **- Joachim Rahmfeld & Mark Butler**
A5: Google Colab Notebook

1. Introduction

With growing generative AI capabilities in the market today, especially through LLMs (Large Language Models), ACME inc. has been looking to invest in developing and productionizing these models in-house. We begin by supporting internal use-cases, as we build up our understanding, and then move for long-term generative AI products that we can ship to customers.

To begin with, we have created a proof-of-concept for an internal question-answering application that can support engineering / research as well as marketing divisions with answering queries on LLMs and GenAI. Thanks to both departments, we were able to receive and curate a list of "gold standard answers" on 75 questions. Each question-answer pair contains an ideal response engineers would need, and an ideal one that marketing would need. Marketing responses generally tend to be more concise and less technical than engineering responses. We use these responses as a gold validation set to measure how well the PoC performs.



2. Methodology

2.1 Technical Approach

The PoC involves multiple components that work together to create a RAG. They are displayed in the figure above. RAG chains need a vector store, LLM and prompt. A chain can be thought of as a microservice that enables one kind of RAG pipeline. We use the QDrant in-memory vector store. This will contain vector embeddings of the documents we want to have in context for the application. The LLM used is Mistral7B from Huggingface, and Cohere. We have one prompt per engineering and marketing teams, to enable customized responses.

2.2 Testing and Evaluation

Metrics

We evaluated each iteration of the models, on well-defined research and industry standard metrics. The key idea was to compare the generated responses from the RAG against the gold standard answers for each question. This comparison needs to account not only for similar words in both responses, but also the order and semantics (meaning) between texts. As such, the following metrics were considered:

Metric	Metric Methodology	Advantages in Q/A	Potential Issues in Q/A
BLEU	Compares n-grams of the model's output with reference text, calculating precision scores for matches.	Quantifies how much the generated text matches a reference text.	Can miss semantic accuracy if the phrasing differs from the reference.
Rouge-L	Measures the longest common subsequence between the generated text and the reference, focusing on recall.	Good for assessing the inclusion of essential content from the reference.	Less sensitive to phrasing and order; may overlook fluency issues.
BERT Score	Uses the RoBERTa model to embed and compare tokens of the model output and reference text, focusing on context.	Consider semantic meaning more deeply than n-gram matching.	Computationally intensive and can be slower to compute.

Note that BLEU and Rouge-L are algorithmic approaches, whereas BERT Score uses a pre-trained BERT Model. As a stand-in for qualitative checking, the BERT Score offers a quantitative approach that still captures qualitative nuances. A fourth metric was considered which involved re-using the embeddings model used to generate the vectorstore to embed the results and compute a cosine similarity between them. This was discarded since BERT performed better at recognizing semantics.

Parameters of Experiment

In order to run experiments and evaluate model performance, we have the following tweakable components:

- Two LLMs (Mistral7B vs Cohere)
- Model Variation (Baseline, Prompt Engineering, Others…)
- Three Chunk Size of Vector Store (Number of Characters per split - 64,128,256). There is an additional overlap component that went unused, that defines how many characters should overlap between sequential splits of texts.
- Two Validation Sets: Engineering + Marketing
- Three Metrics: BLEU, ROUGE-L, BERT-Score

Experiment Results Table

By taking all combinations of the above, we arrive at the following experiment results table.

- All metric scores are scaled from range [0-1] to [0-100]. So 0.22 would be 22.
- Baseline represents the actual scores, and all other rows represent the percentage-point deltas compared to the baseline performance. For example, If the baseline was 22, and the delta on Prompt variation was 4, then the result of that metric for that run was 26 (or 0.26). We only show the P.P deltas in the table.

Model	Variation	Chunk	ENG-BLEU	MKT-BLEU	ENG-RL	MKT-RL	ENG-BERT	MKT-BERT
Mistral7B	🟢 Baseline	128	8	7	22	22	87	87
	▲ Prompt	64	2	4	4	8	1	2
		128	3	6	4	11	2	3
		256	3	7	5	11	2	3
Cohere	🟢 Baseline	128	6	4	20	15	86	85
	▲ Prompt	64	0	2	4	6	1	2
		128	3	5	6	14	2	4
		256	2	2	4	4	1	2

Comments on Parameter Tuning & Experimental Runs

Chunk size variation (by recursive character text splitting) was an important factor since it decides the dimensionality and context of vectors in the vectorstore, thereby providing partial or whole information to the model using the Langchain retriever. As such, we iterated on 3 variants of chunksize.

The main variation from baseline was to improve the LLM prompt passed to the models. One prompt each for engineering and marketing was used, resulting in one lang-chain for each. By analyzing the distribution of tokens in the gold validation answers, we determined that marketing on average requires answers not more than 50 words, and engineering can deal with more technical answers at almost 100 words. These word limits, together with additional context on the audience, were incorporated into the prompt to gain performance.

Experiment Results:

- Prompt Engineered RAG chains perform better than the baseline every time.
- Mistral's delta on 128 and 256 chunks are comparable, with 256 parameterized chunks offering the best performance based on selected metrics. Cohere saw best performance on 128 parameterized chunks, with 256 being detrimental in relation to 128, but still better than baseline.

3. Results & Findings

3.1 Proof of Concept Functionality

The PoC has successfully demonstrated the core idea of using a RAG system to answer domain-specific questions by leveraging different sources of information and LLMs

- In order to use language models for a majority of document retrieval and up-to-date question answering use-cases, we as a company don't need to train LLMs from scratch. We can instead use RAG pipelines by preloading them with essential documents so that a pre-trained LLM has the required context without expensive training effort and time.

- Despite low BLEU scores, we got satisfactory LCS (longest common subsequence) overlap using ROUGE-L, for both engineering and marketing aspects, while also receiving BERT scores reaching 90 - indicating very high semantic similarity between generated responses and the gold validation set.

3.2 Lessons Learned, Challenges & Limitations

RAGs have a tradeoff between quick iterative update of models with latest information, while not spending as much time retraining existing models with new information. RAGs have great potential utility, but their ability to capture nuances from new documents is also limited after a point and the performance plateaus. Larger chunks improve context but may reduce the number of relevant documents retrieved. Different stakeholder groups (engineering vs. marketing) require tailored information delivery, stressing the importance of understanding end-user needs, and expanding chains for new audiences. The iterative nature of model optimization highlights that large gains often come from cumulative small tweaks showing measurable improvement in answering capabilities through iterations in system configuration.

3.3 Recommendations

It is recommended to begin setting up a RAG pipeline for internal question answering use-cases as part of ACME's tech stack. This will be a cross-functional effort and will require support from various stakeholders. The return on investment can translate to higher productivity, improved LLM capabilities that can convert to skill sets that allow us to develop external user-facing GenAI products with in-house capabilities.

Major risks to keep in mind as we move forward are that the performance of RAGs are highly dependent on the quality and freshness of the data provided as context, the prompts used and the LLM under the hood. For instance, Mistral is not explicitly trained to not provide unsafe outputs - so figuring this out is key before making any LLMs user facing.

Proposed Design

The proposed system can be structured as a robust data pipeline designed to efficiently manage and utilize diverse data sources:

Document, Context & Prompt Management

- **Index of Allowed Context:** Maintain a dynamic index that manages access to relevant documents. This index allows the pipeline to refresh the vector store as needed or according to a predefined schedule.

- **Audience-Feature-Store (AFS):** Link documents relevant to specific audience groups to this store, facilitating tailored content delivery based on audience characteristics.

- **Index of Prompts:** Establish a repository where new prompts can be added or existing ones modified. These prompts are utilized by the pipeline in subsequent executions to ensure relevance and accuracy.

User Interaction and Feedback

- **Metrics Log**: Record metrics from every user interaction to track performance and user satisfaction.

- **Continuous Feedback Mechanism**: Implement a system where user feedback is continuously gathered and utilized not only to refine prompts and documents but also to potentially recalibrate and customize the evaluation metrics.

Infrastructure and Scalability

- **Vector Database:** Transition from an in-memory vector store to a more scalable vector database, suitable for handling larger datasets and complex queries.

- **Stable Compute Environment:** Ensure the pipeline has a stable compute environment for refreshing its context regularly.

- **User Interface API**: Develop an API that facilitates seamless interaction between the end-user and the pipeline, ensuring a user-friendly experience.