Evaluating RAG LLMs for Response Quality

EDS 6397 - Natural Language Processing, Spring 2025
Team 3: Quang Do, Ruthwik Reddy Karna, Gopi Trinadh Maddikunta

Abstract

LLMs have taken off in recent years for their ability to answer questions quickly and, usually, accurately. However, the main pitfall of LLMs is their inability to update their knowledge with changes in documentation or training data, as their memory is static. Moreover, LLM usage in enterprise necessitates knowledge of proprietary information, which the LLM was not trained on. Therefore, queries relating to this proprietary information might often lead to hallucinatory answers. The current best method to imbue LLMs with this proprietary or updated knowledge is through RAG. Our project aims to evaluate the answer quality and retrieval quality of different LLMs under the same retrieval pipeline, motivated by the fact that LLM benchmark scores often scale with the number of parameters in the LLM. We used 80m, 500m and 1b parameter models (Flan-T5 Small, Qwen2.5, Llama 3.2, respectively) Our findings show that the largest LLM used (Llama 3.2-1B-Instruct-QLORA-Int4-EO8) performed the best in comparisons to smaller models, under our answer relevance, faithfulness, context precision and context recall metrics, with considerable decreases in answer relevance and faithfulness as we decrease the parameters of the models.

Introduction

Large language models (LLMs) have transformed the way humans seek out information. For complicated queries, or for information not easily attainable through Google Search, ChatGPT and its peers have taken center stage for their conciseness and convenience. Many companies are now using LLMs (through Copilot, Cursor, or ChatGPT) to enhance the workflow of their engineers and employees; instead of spending their time Googling, a simple prompt to a local or cloud based LLM can answer most of their non-proprietary related queries.

However, despite the rise of LLMs, there are still a few concerns. Namely, the information traditional non agentic LLMs contain are limited to what they are trained on; this means their “memory” is static, and does not adapt to any changes or new information that comes from updates in documentation or otherwise. Moreover, LLMs do not contain any proprietary data. This is especially noteworthy for the enterprise use case: if you’re an engineer looking for any proprietary information, whether its a specification, tolerance, or architecture, an LLM will not be able to find this information, as it was not trained on proprietary data. In an attempt to fulfill the query, LLMs might hallucinate as well [1], which is dangerous, as, without proper verification, the hallucinated output might appear reasonable and even true, depending on the hallucination. Obviously, using this hallucinated information is unadvisable.

The problem then becomes the following: “How can we imbue LLMs with knowledge of proprietary, internal documents?” Fine-tuning is not optimal here because we would have to fine-tune our model on any new corpora any time we wished to update our knowledge base [3]. Additionally, what if we wish to retrieve the exact document containing the relevant information? Fine-tuning does not allow for this. The best and most common solution to this problem is retrieval augmented generation (RAG hereafter), developed in 2020 [2].

RAG, at a very-high level, is a simple four step process.

1. Documents that you wish to retrieve information from are processed and stored in a database as embeddings; such a database is known as a vector database.
2. Relevant information is retrieved by analyzing the similarity between a given query and the documents in the database.
3. Retrieved relevant information is added to the input prompt's context, along with the question, and sent to the LLM.
4. A relevant, informed response is generated.

RAG is dependent on a LLM to generate a response given the context and question. The natural next step, then, is to evaluate the response generated by the LLM, as well as the documents retrieved by the retrieval system and datastore, to determine the quality of the response and retrieved documents. Additionally, it is well known that LLMs perform better on benchmarks as a function of their parameters [8], but does this hold for the quality of RAG responses as well? This question is the focus of our project.

Dataset

This project employs the publicly available neural-bridge/rag-dataset-12000 dataset (the dataset, hereafter), found on the Hugging Face platform [4]. Designed for RAG architecture evaluation, this dataset simulates an open-domain question answering task, closely approximating real-world language understanding tasks. The size and organization of the dataset make it particularly well-suited for evaluating transformer-based model performance in a RAG pipeline.

The data includes 12,000 samples split into training and test sets. Each sample consists of three components: (i) a question—a natural language question that must invoke external knowledge, (ii) a context—a paragraph or passage of context information required to answer the question, and (iii) an answer—a ground truth response that has been hand-annotated. Such a structure can be used with both retrieval and generation aspects of the RAG architecture as well as simple performance comparison across models on the basis of the information retrieved.

The dataset was selected driven by three principles. Firstly, it includes a wide range of topics and question types, which enables effective generalization and exposes each model to different levels of semantic complexity. Secondly, the dataset supports RAG evaluation metrics like faithfulness, answer relevance, context precision, and context recall, which makes it suitable for benchmarking against applications like RAGAS. Third, it provides a balance between short and long contexts, hence questioning the effectiveness of the retrieval and chunking methods.

As an added point of convenience, the dataset is already thoroughly cleaned, every (question, answer, context) entry has every index filled with relevant text. It's also worth noting here that we did not take any additional measures to "clean" the context (or any part of the tuple for that matter), for a few reasons. Firstly, the context is already clean, in that it is not missing for any entries, and is relevant for all entries. We can assure ourselves of the former, because the question and answer were generated by GPT 4 in the dataset[4], from the context. Secondly, the diversity of the context relies on the fact that it is taken from different sources, and that it varies in levels of cleanliness; some samples are from text, others are from screenshots, where the text has been scraped. Leaving the context as it is (from Falcon-RefinedWeb) [4] allows us to more concretely evaluate metrics; if the context was cleaned, it would not be representative of the text that results from web or document scraping (text that would then go into a vector database, inevitably) and give an unfair advantage to the generator, causing it to generate much cleaner responses due to the clean context, in comparison to the standard context that comes with the dataset.

In order to overcome the input length constraint of Flan-T5 small (512 tokens), each context passage was broken down into small overlapping segments by utilizing the RecursiveCharacterTextSplitter in the LangChain library [5]. A chunk size of 500 characters and overlap of 50 characters were utilized to ensure that no informative content was sacrificed while segmenting. This step was not required for all models. It is seen as an accommodation step for models whose context sizes are too small to store the context natively in their prompts.

Each resultant piece was then embedded into a high-density vector representation using the SentenceTransformers pretrained all-MiniLM-L6-v2 model [6]. This embedding allowed the building of a FAISS index supporting high-speed semantic search during retrieval. After our dataset is transferred to the vector database, it is ready for RAG. At test time, every question in the test split was input into this index to retrieve the top-k (k in our scenario is always 1, as each question contains exactly 1 relevant piece of context; see the dataset for more information) contextually salient pieces, which were then input—along with the question itself—into the language model for generation.
The rag-dataset-12000 corpus is particularly appropriate for this research as it reflects the complexity and vagueness of actual queries. The dataset permitted us to test three instruction-tuned language models—Flan-T5 Small, Qwen2.5-0.5B-Instruct-GPTQ-Int4, and Llama-3.2-1B-Instruct-QLORA_INT4_EO8 —under identical retrieval conditions. This permitted us to observe the impact of model architecture and parameter size on performance metrics, gaining insights into each model's strengths and weaknesses under a RAG-based configuration.

It's worth noting that the structure of this dataset (1 relevant document per question and answer pair), does not necessarily necessitate the use of a vector database, as the relevant document and question could just as easily be injected into the prompt, but as an industry-relevant exercise, our team decided to implement the vector database, as it is much more likely to be encountered in an industry setting as opposed to the edge case of our dataset not necessitating one.

Methodology

After the context from the dataset is embedded and stored in the database above, we can begin the construction of our RAG pipeline. Using HuggingFace, we can instantiate a language model pipeline as seen in figure 1.

```
# Make pipeline
hf_gen = pipeline(
    "text-generation",
    model=model_id,
    tokenizer=tokenizer_id,
    torch_dtype=torch.float16,
    device=0,
    max_new_tokens=1024,
    batch_size=20,
    temperature=0.01,
    do_sample=True,
)
hf_gen.tokenizer.pad_token = hf_gen.tokenizer.eos_token

# Wrap it in LangChain's LLM interface
llm = HuggingFacePipeline(pipeline=hf_gen, model_kwargs={"stop": ["}\n```", "}\n\n", "}\nHere is"]})
```

Figure 1: Generic Pipeline Instantiation

The temperature and max new tokens are hyperparameters, and were tested using a small sample of our test data to maximize the faithfulness and answer relevance metrics. We then wrap it in LangChain's HuggingFaceLLM wrapper to turn it into, for lack of a better word, a usable LLM. This pipeline will be used to generate responses to our queries, based on the provided context, and can now be used to generate new responses.

The construction of the RAG pipeline is rather straightforward, and is little more than a call to Langchain's RetrievalQA function. We must first construct a retriever for our vector database, where, notably, we retrieve only one document from the vector database. This is an artifact of our dataset, where each [context, answer, question] tuple only contains one context for each question. We are not sure if the contexts overlap (if the context of one question can answer another, in addition to the one it actually answers), but the dataset does not suggest this in the documentation or structure. After we build the retriever, we can then instantiate a RetrievalQA object, which will use the LLM and the retriever to retrieve relevant contexts given our query. The steps described above can be seen in figure 2; this implementation is fairly straight forward.

```
# Build retriever
retriever = vectorstore.as_retriever(search_kwargs={"k": 1})

# 4. Now create the RetrievalQA chain
qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",                # or "map_reduce" / "refine" if you prefer
    retriever=retriever,
    return_source_documents=False
)
```

```
evice set to use cuda:0
```

```
# ----------------------------------------------------
# 3.2 Test it on the first user_input
# ----------------------------------------------------
res = qa_chain({"query": test_data["question"][0]})
```

Figure 2: Retriever, qa_chain, implementation and example usage.

It's worth noting that res is the entire output from the retrieval chain; that is, it is a .json containing the sent query (from the dataset) and the result, which contains the query from the dataset, as well as the retrieved context and the LLM's output. An example of the result from the above code can be seen in figure 3.

```
WHOLE RESPONSE:
{'query': 'What is the latest version of Scorpion Solitaire?', 'result': "Use the following pieces of context to answer the q
uestion at the end. If you don't know the answer, just say that you don't know, don't try to make up an answer.\n\nScorpion S
olitaire\n- User\n- -\n-\n- Insufficient votes\n- Softonic\n- 6\n- Not bad\n- Not bad\n- Your rating:\n-\nYour rating has bee
n saved\nOops, something's gone wrong. Try again.\n- License:\n- Free\n- Language:\n- OS:\n-\n- Latest version:\n- 1.1 17/02/
06\n- Last month's downloads:\n- 41\n- -...\n- PocketBalone\n- Billiard Master\n- Chess\n- ...\n- 21\nScorpion Solitaire\nSof
tonic - Top Downloads\nTop Downloads\n- Pocket Uno\nPlay the classic card game on your Pocket PC\n- Multiplayer Championship
Poker -...\n- PocketBalone\n- Billiard Master\n- Trivial Pursuit\n- ...\n- 21\nScorpion Solitaire.\n\nQuestion: What is the l
atest version of Scorpion Solitaire?\nHelpful Answer: According to the Softonic website, the latest version of Scorpion Solit
aire is 1.1, released on 17/02/06."}
```

Figure 3: output of print(res), highlighted for clarity

The result entity within the .json (the output of res is in a .json format) will need to be parsed to extract the actual LLM response, but this is trivial, as its answer will always follow "Helpful Answer" or "Answer" Critically, res, although not printed above (because it is a Document data type), also contains the source documents, which are the retrieved documents from the database that were inserted in the prompt to provide the LLM with context. After querying the database, and getting our generated response from the LLM, we now have everything we need to begin evaluating the response on context precision, context recall, faithfulness, and answer relevance. A diagram of the flow of information from the database to our LLMs are presented below in figure 4.

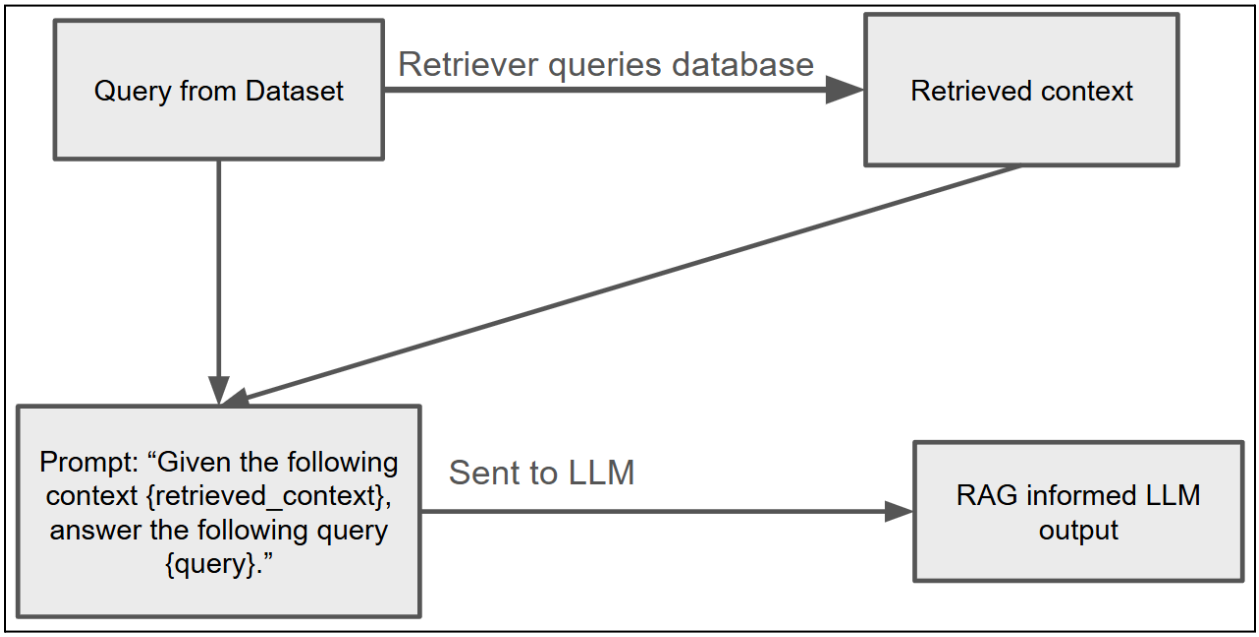

Figure 4: Database to LLM output diagram

Context precision and context recall can both be calculated using Ragas [7], which is an open-source LLM evaluation library. Critically, context precision and context recall do not need the intervention of an LLM to evaluate the generated response. Instead, both context precision and context recall utilize similarity based methods (Levenshtein distance, by default) to evaluate these metrics.

Context precision is effectively a measure of how much retrieved context is relevant, conceptually, it is the fraction of the overlapping info within the total amount of info in the reference context. Context recall is a similar metric, measuring the number of claims within the reference that were supported by the retrieved context over the total number of claims in the reference. They are defined in figure 5 and 6, respectively.

$$\text{Context Precision@K} = \frac{\sum_{k=1}^{K} (\text{Precision@k} \times v_k)}{\text{Total number of relevant items in the top } K \text{ results}}$$

$$\text{Precision@k} = \frac{\text{true positives@k}}{(\text{true positives@k} + \text{false positives@k})}$$

Figure 5: Definition of context precision

$$\text{Context Recall} = \frac{\text{Number of claims in the reference supported by the retrieved context}}{\text{Total number of claims in the reference}}$$

Figure 6: Definition of context recall

It's worth noting that the precision in figure 5 is defined, practically, as the ratio between the number of relevant chunks to the total number of chunks; that is, the fraction of the relevant retrieved references over all retrieved references [7].

Faithfulness and answer relevance require LLM intervention to properly evaluate responses to construct the two prior metrics. These metrics are defined in figure 6 and 7, respectively.

$$\text{Faithfulness Score} = \frac{\text{Number of claims in the response supported by the retrieved context}}{\text{Total number of claims in the response}}$$

Figure 6: Definition of Faithfulness; not to be confused with number of claims in reference (context recall)

$$\text{answer relevancy} = \frac{1}{N}\sum_{i=1}^{N} cos(E_{g_i}, E_o)$$

$$\text{answer relevancy} = \frac{1}{N}\sum_{i=1}^{N} \frac{E_{g_i} \cdot E_o}{\|E_{g_i}\|\|E_o\|}$$

Where:

- E_{g_i} is the embedding of the generated question i.
- E_o is the embedding of the original question.
- N is the number of generated questions, which is 3 default.

Figure 7: Definition of Answer Relevance

The required usage of LLMs for answer relevance is obvious; the question generation required is really only possible with an LLM. The LLM intervention is required for faithfulness because we need to manually extract the claims from within the response for an accurate evaluation, in contrast to the under-the-hood Levenshtein distance context recall uses. However, RAGAS' default LLM of choice is OpenAI's GPT models. Though much more performant, this is expensive. Therefore, it is economically efficient to evaluate these metrics locally.

To this end, custom prompts were developed for both faithfulness and answer relevance to be used upon a judge LLM, for all models, accommodating their varying levels of intelligence. The flow of metric calculation using our local judge LLM (Llama 3.2-3B Instruct) is seen in Figure 8.

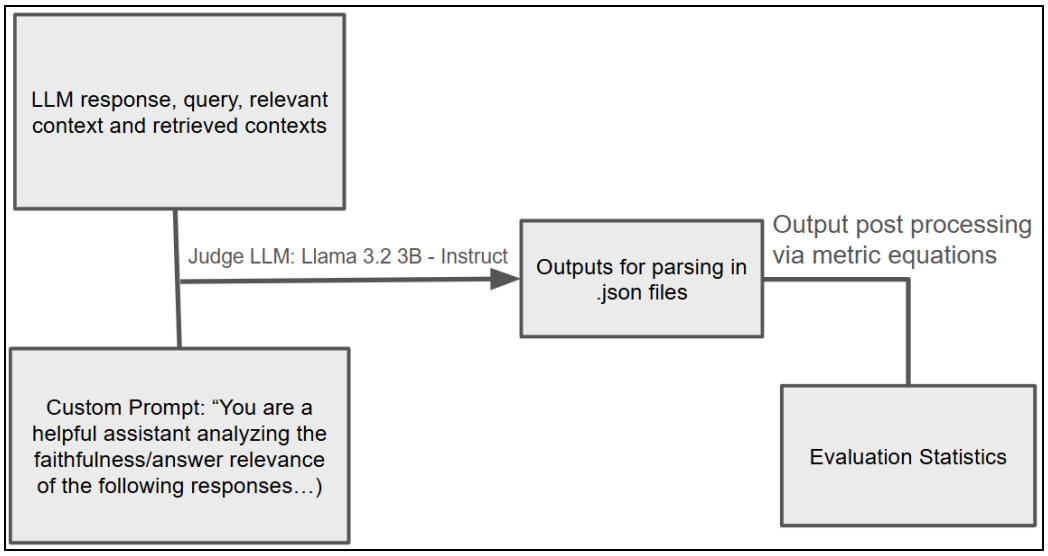

Figure 8: Data flow from RAG output to outputs

The LLM is used to generate the questions for answer relevance, and also identify the claims in the response, and which claims are supported by the retrieved context, which is required by faithfulness. These questions and identified claims are then post-processed using RAGAS' formulas, specifically, answer relevance was calculated as the average of the cosine similarity of the generated questions and the original question, after both were embedded using our embedding model. Faithfulness was calculated simply by dividing the number of claims in the response supported by the retrieved context divided by the total number of claims in the response.

After the average metric scores are calculated, we can proceed with the calculation of the standard deviation and 95% confidence interval, the latter of which we did using a t-distribution and the standard error of the mean for our samples, as we cannot guarantee normality.

Results and Discussion

A sample of the LLM output for faithfulness, specifically, is included in figure 9. Note that the RAG response itself is located in "response," stripped of the question and retrieved context. The question in figure 9 can most likely be inferred, but asked about the latest version of Scorpion Solitaire.

```
output:
{
  "response": "According to the Softonic website, the latest version of Scorpion Solitaire is 1.1, released on 17/02/06.",
  "context": "Scorpion Solitaire\n- User\n- -\n-\n- Insufficient votes\n- Softonic\n- 6\n- Not bad\n- Not bad\n- Your ratin
g:\n-\nYour rating has been saved\nOops, something's gone wrong. Try again.\n- License:\n- Free\n- Language:\n- OS:\n-\n- Lat
est version:\n- 1.1 17/02/06\n- Last month's downloads:\n- 41\n- -...\n- PocketBalone\n- Billiard Master\n- Chess\n-...\n- 21
\nScorpion Solitaire\nSoftonic - Top Downloads\nTop Downloads\n- Pocket Uno\nPlay the classic card game on your Pocket PC\n-
Multiplayer Championship Poker -...\n- PocketBalone\n- Billiard Master\n- Trivial Pursuit\n-...\n- 21\nScorpion Solitaire.",
  "chain_of_thought": "1. Analyzed Response ONLY: Identified one atomic claim directly stated: 'the latest version of Scorpio
n Solitaire is 1.1, released on 17/02/06'. 2. Finalized `response_claims` with this one claim. 3. Analyzed Context: Identifie
d claims about user interface, insufficient votes, Softonic rating, free license, latest version, and downloads. 4. Identifie
d `context_only_claims`: Compared context claims (Step 3) to response claims (Step 2). Found claims about user interface, ins
ufficient votes, Softonic rating, free license, latest version, and downloads are unique to the context. 5. Verified `respons
e_claims`: Claim 1 ('the latest version of Scorpion Solitaire is 1.1, released on 17/02/06') is supported by context. 6. Gene
rated Justifications: Created justification for the response claim based on context verification. Keys exactly match `respons
e_claims`. 7. Counted `response_claims`: Total=1, Supported=1. Prepared count justifications. 8. Formatted JSON output.",
  "response_claims": [
    "the latest version of Scorpion Solitaire is 1.1, released on 17/02/06"
  ],
  "context_only_claims": [
    "Scorpion Solitaire\n- User\n- -\n-\n- Insufficient votes\n- Softonic\n- 6\n- Not bad\n- Not bad\n- Your rating:\n-\nYour
rating has been saved\nOops, something's gone wrong. Try again.\n- License:\n- Free\n- Language:\n- OS:\n-\n- Latest versio
n:\n- 1.1 17/02/06\n- Last month's downloads:\n- 41\n- -...\n- PocketBalone\n- Billiard Master\n- Chess\n-...\n- 21\nScorpion
Solitaire\nSoftonic - Top Downloads\nTop Downloads\n- Pocket Uno\nPlay the classic card game on your Pocket PC\n- Multiplayer
Championship Poker -...\n- PocketBalone\n- Billiard Master\n- Trivial Pursuit\n-...\n- 21\nScorpion Solitaire."
  ],
  "supported_response_claims": [
    "the latest version of Scorpion Solitaire is 1.1, released on 17/02/06"
  ],
  "unsupported_response_claims": [],
  "justifications": {
    "the latest version of Scorpion Solitaire is 1.1, released on 17/02/06": "in context - context states the latest version
and release date"
  },
  "num_supported_response_claims_justification": "Derived by comparing the 1 claim identified strictly from the response agai
nst the context. 1 claim was found to be fully supported.",
  "num_supported_response_claims": 1,
  "num_total_response_claims_justification": "Derived by breaking down the response ONLY into distinct factual statements. 1
atomic claim was identified.",
  "num_total_response_claims": 1
}
```


Figure 9: Sample Judge output from parsing RAG response

Table 1 below documents the results of the metrics we evaluated for each model, as well as their 95% confidence interval and standard deviation.

Model	[Context Precision ± 95% Confidence Interval, Standard Deviation]	[Context Recall ± 95% Confidence Interval, Standard Deviation]	[Faithfulness ± 95% Confidence Interval, Standard Deviation]	[Answer Relevance ± 95% Confidence Interval, Standard Deviation]
Llama-3.2-1B-Instruct-QLORA_INT4_EO8	[0.826667 ± 0.0152, 0.3786]	[0.826667 ± 0.0152, 0.3786]	[0.7206 ± 0.0235, 0.3816]	[0.8353 ± 0.0167, 0.2894]
Flan-T5 Small	[0.652389 ± 0.0192, 0.5428]	[0.523219 ± 0.0182, 0.4523]	[0.4522 ± 0.0267, 0.4854]	[0.5988 ± 0.0342, 0.4023]
Qwen2.5-0.5B-INstruct-GPTQ-Int4	[0.826667 ± 0.0152, 0.3786]	[0.826667 ± 0.0152, 0.3786]	[0.6273 ± 0.0263, 0.4233]	[0.7587 ± 0.0212, 0.34]

Table 1: Models and their respective performance metrics; highest scores are in green.

As predicted, the largest model (Llama 3.2, with 1B parameters) performed the best in faithfulness and answer relevance, with a noticeable decrease in the faithfulness and answer relevance and we decrease our parameters. These metrics, obviously, are dependent on the quality of the judge, as stated prior, but we believe this trend generally holds even with an intelligent judge.

What's interesting to note is that the context precision and recall scores for both the Qwen2.5 model and the Llama 3.2 model are the same. This is because, unlike the Flan-T5 small model with a context length of 512 tokens, the context lengths of the Qwen2.5 and Llama 3.2 models were much higher, which meant we did not need to "chunk" the documents into smaller sections to store in our vector database for retrieval and insertion into prompts. Qwen2.5 and Llama 3.2 were both able to use the entire document stored in the database for retrieval and prompt context. As such, due to our consistent retrieval methods across all of our models, the scores are exactly the same.

However, as mentioned prior, Flan-T5 small required document chunking to properly process the contexts. Due to the methods of determining context precision and context recall (average Levenshtein distance between the retrieved and the reference context), Flan-T5 Small's context precision and recall scores were significantly lower, as chunking resulted in fragmented information within the chunked documents, which led to an increase in the Levenshtein distance between the chunked contexts and the reference contexts, leading to lower precision recall and precision metrics. Moreover, conceptually, it is most likely more difficult to write a single coherent response from several small chunks as opposed to the longer, larger chunks of contexts retrieved by our more capable models, which most likely further exacerbated the poor response quality of Flan-T5 small. The presence of chunking most likely is a confounding variable for all metrics, although more evident on the context precision and recall metrics.

It is also likely that our K=1 (one context, per document, per question) did not thoroughly test our retrieval and generation methods. Increasing the amount of context per question and answer through a larger dataset or other method is within the scope of future work for this project to get an even overview of our pipeline and generative capabilities.

Investigating the effects of using quantized models versus unquantized models is also in the scope of future work, though we don't anticipate the effects of quantization to drastically impact the generation metrics.

Conclusion

In an unsurprising turn of events, the model with the largest parameters performed the best in our analysis. However, there are a few caveats to this. Our sample size was rather small, using only the test data from our dataset instead of the entire dataset itself. In reality, we could have also evaluated on the "train" set, because we weren't doing any fine-tuning, but limitations in our computational resources and time constraints did not allow us to do so. Evaluating on the rest of the dataset would give us more confidence in our findings, but this is beyond the capabilities of our group's computational resources. For reference, our test dataset ended up being 2.4k [context, question, answer] tuples, while the training data consisted of 9.6k [context, question, answer] tuples.

Given the choice between these three models, and the metrics calculated in this paper, Llama 3.2 is clearly the preferred model when it comes to LLM generated RAG responses. Flan-T5 small suffers greatly due to its context length, which limits the amount of contexts it can contain from our vector database within its prompts. This problem becomes even more obvious in its low faithfulness and answer relevance scores; these context chunks most likely make it more difficult to form coherent responses, as well,

though this is left as future work. Qwen2.5 is better than Flan T5-small, but still isn't as good as our Llama 3.2 models.

Moreover, we found that the judge model had a very profound impact on our ability to properly evaluate our responses. We parsed LLM outputs in .json files, which means that, if the LLM could not properly produce these .json, we would fail to properly evaluate the responses generated by our LLMs. This problem is solved by using more robust prompts and more robust models, but the former imputes significant time and computational costs associated with processing large inputs (prompts) and outputs, and the latter imputes significant financial costs, as the strongest models that could perfectly create our expected .json files are often too large to host locally, and require remote hosting via Google Colab or using APIs such as Gemini or OpenAI. It's unlikely that a 3B parameter model perfectly evaluated our metrics, so using a more capable LLM in a future work might provide more accurate values for these metrics.

It's also worth noting that the prompts for judge model were tailored to maximize the compatibility of the response with our native writing .json output parsers; different prompts may lead to varying answers, as is the nature with prompts, though we suspect our prompts did a fairly good job of giving us accurate answers. Moreover, it's probably worth evaluating more metrics than just faithfulness and answer relevance in future works. Answer correctness is one that comes to mind immediately, which measures the accuracy of the response with the ground truth.

Lastly, our usage of RAGAS in this project was significantly less than expected. Originally, we had planned to evaluate all four of our metrics using RAGAS' evaluate() function, however, due to RAGAS' .json parser failing on the outputs judge models up to 8B parameters, we were forced to write our own prompt and use a local LLM to evaluate the quality of our responses. This was a headache, and introduced significant time and computation costs to our project. It is likely we would have been able to evaluate more samples in our given timeframe had RAGAS worked as intended with our local LLMs.

References

[1]L. Huang *et al.*, “A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions.” Available: https://arxiv.org/pdf/2311.05232 (accessed Apr. 17, 2025)

[2] P. Lewis et al., Retrieval-augmented generation for knowledge-intensive NLP tasks | proceedings of the 34th International Conference on Neural Information Processing Systems, https://dl.acm.org/doi/10.5555/3495724.3496517 (accessed Apr. 17, 2025).

[3] “Rag vs. fine-tuning,” Red Hat - We make open source technologies for the enterprise, https://www.redhat.com/en/topics/ai/rag-vs-fine-tuning (accessed Apr. 17, 2025).

[4] neural-bridge, “rag-dataset-12000,” Hugging Face, https://huggingface.co/datasets/neural-bridge/rag-dataset-12000 (accessed Apr. 18, 2025).

[5] LangChain, “Text Splitters,” LangChain Documentation, https://docs.langchain.com/docs/modules/data_connection/document_transformers/text_splitters (accessed Apr. 18, 2025).

[6] Sentence-Transformers, “all-MiniLM-L6-v2,” Hugging Face, https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2 (accessed Apr. 18, 2025).

[7] “✨ introduction,” Ragas, https://docs.ragas.io/en/stable/ (accessed Apr. 26, 2025).

[8] Chuhan Wu LLM Researcher &Ruiming Tang LLM Researcher, Performance law of large language models, https://arxiv.org/html/2408.09895v2 (accessed Apr. 26, 2025).