DRAFT : Development and Evaluation of Finance RAG Using LlamaIndex

Ankan Kumar Ray
Under the guidance of Dr. Tanmay Sen
Indian Statistical Institute (ISI), Kolkata

1 Introduction

Large Language Models (LLMs) have revolutionized natural language processing by learning rich representations of text .

Historical Context of Large Language Models (LLMs)

Large Language Models (LLMs) are a class of deep neural networks trained to understand and generate human language. Their development marks a significant shift in natural language processing (NLP), moving from rule-based and shallow statistical models to scalable, data-driven architectures capable of learning rich representations from raw text.

The term **'large'** in LLM refers to both the size of the model (typically billions of parameters) and the vast training corpora used to build it. A fundamental breakthrough occurred with the publication of the *Transformer* architecture in the 2017 paper *"Attention Is All You Need"* by Vaswani et al. This architecture introduced the self-attention mechanism, enabling models to efficiently capture long-range dependencies without recurrence.
Following the transformer, the first generation of pre-trained LLMs emerged:

- **GPT (Radford et al., 2018)[OpenAI]**: The first *Generative Pretrained Transformer*, trained using left-to-right next-word prediction in large text. It demonstrated the power of unsupervised pretraining followed by task-specific fine-tuning.
- **BERT (Devlin et al., 2018)[Google AI]**: *Bidirectional Encoder Representations from Transformers* introduced masked language modeling and bidirectional

attention. BERT set new benchmarks on a range of NLP tasks and popularized encoder-based LLMs for classification.

- **GPT-2 and GPT-3**: Significantly scaled up the number of parameters (up to 175 billion in GPT-3) and demonstrated few-shot and zero-shot learning capabilities—models could solve tasks by simply being prompted, without explicit fine-tuning.

LLMs transitioned from research to real-world impact with the release of **ChatGPT** in 2022, built on GPT-3.5 and later on GPT-4.

In parallel to closed-source development by firms like OpenAI and Google etc. , open-source ecosystems also grew. **LLaMA** by Meta, **Mistral**, **Bloom**, and **Qwen** emerged as powerful alternatives to proprietary models, allowing researchers and developers to build domain-specific LLM applications. Many of these models are used in **Retrieval-Augmented Generation (RAG)** pipelines which is what we will look at in this project.

Modern LLMs are characterized not only by scale but also by modular design, extensibility, and fine-tuning techniques that allow them to be adapted to new domains and languages.

However, even the most powerful LLMs are known to hallucinate—generating content that is fluent but factually incorrect. To mitigate this, the field has shifted toward RAG, where information retrieval modules supplement LLMs with external context, grounding their outputs in verifiable data.

This project evaluates a modern RAG pipeline that uses the open-source **Qwen3-1.7B** model as the generative engine, it's built upon the **LlamaIndex** framework for unifying document retrieval, structuring, and querying in one environment. Evaluation is performed using both lexical and semantic metrics: **BLEU, ROUGE, BERTScore**, and internal evaluators provided by LlamaIndex for faithfulness and relevance.

Historical Context of Submodules

Qwen3-1.7B: The Qwen series of language models, developed by Alibaba DAMO Academy, represents one of the most performant **open-source** alternatives to proprietary LLMs like GPT-3. Qwen's multilingual understanding and instruction following position it as a viable model for chat, QA, and summarization tasks.

HuggingFace

From Trnsformers module:

1. **AutoModelForCausalLM**: Create an instance of the pre-trained LLM model in the system locally using the .from_pre-trained() class method. This is essential as we need the generative LLM to be local , so that information does not leave the system, ensuring **data privacy**.
2. **AutoTokenizer**: To load the tokenizer associated with the pretrained LLM, essential for converting text into tokens that the model can understand, and for converting the model's output tokens back into human-readable text. It also uses the from_pretrained() class method to ensure compatibility with the chosen model.

LlamaIndex (formerly known as `GPT Index`) is an open-source framework designed to facilitate the integration of external data sources—structured and unstructured—into LLM-powered pipelines. It was introduced by in 2022 with the aim of enabling flexible and modular construction of Retrieval-Augmented Generation (RAG) systems.

At its core, LlamaIndex acts as a mediator between raw data (PDFs, text files, web pages, databases) and large language models, providing tools to structure, store, and retrieve relevant information for contextualized question answering. Its modularity allows developers and researchers to plug in various backends for embedding models, vector storage, and LLM inference, making it ideal for experimentation and production systems.

The main submodules of LlamaIndex used in this project are as follows:

1. **SimpleDirectoryReader**: A data loader that ingests text from local directories.
2. **TokenTextSplitter** : A node parser. It chunks, and stores documents as nodes in memory. Each node retains metadata that supports future filtering and tracing.
3. **VectorStoreIndex**: This module builds a dense vector index from document embeddings using a configurable embedding model. Internally, it often uses `FAISS` (Facebook AI Similarity Search) to allow fast approximate nearest-neighbor retrieval. The `VectorStoreIndex` allows efficient similarity search over document chunks.
4. **Pipeline or HuggingfaceLLM wrapper:** See Documentation

   LLamaindex function to wrap different LLM entities together and set up of the LLM model . Some of the important parameters are:

- `context_window` (`int`): Maximum number of tokens the model can consider in its input context.
- `device_map` (`str`): For set up of device map, enables GPU access.
- `generate_kwargs` (`dict`):
  Keyword arguments for the `generate()` method, the crucial ones are:
  - `temperature`: The temperature controls the randomness or creativity of the text generated. It defines how likely it is to choose tokens that are less probable. A temperature of 0 generates the same response every time because it always chooses the most likely word.
  - `topK`: Controls exactly how many tokens the LLM can consider. If you change its value to 100, the LLM will only consider the top 100 most probable tokens.
  - `topP`: topP, also known as nucleus sampling, is a sampling technique that controls which subset of tokens (the nucleus) the LLM can consider. It will consider tokens until it reaches their cumulative probability.
- `is_chat_model` (`bool`): Whether the model uses a chat-style prompt format.
- `max_new_tokens` (`int`): Maximum number of new tokens to generate in the output.
- `model_kwargs` (`dict`): Additional arguments passed while loading the model, like the **quantization** configuration. Quantization is discussed briefly later.
- `system_prompt` (`str`): prompt that sets the overall context, behavior, or persona for the model's responses. Often it is not directly visible to the end-user.
- `tokenizer_kwargs` (`dict`): Arguments for tokenizer loading (e.g., `use_fast`).

5. **ServiceContext**: A configuration hub that binds the LLM, embedding model, node parser. It enables reproducibility and consistency across query pipelines.

6. **QueryEngine**: The final component that combines retrieved documents, prompt formatting, and the LLM to generate a response. A query engine takes in a natural language query, and returns a rich response. It is built on indexes via retrievers

7. **Retriever**: Retrieves the top-k most relevant chunks or **nodes**, from the index given a query. Advanced retrievers can be layered or reranked using cross-encoders, or fused with keyword and dense retrieval.

SBERT : Unstructured textual data by itself is often quite hard to process. They are not values we can directly process, visualize, and create actionable results from. We first have to convert this textual data to something that we can easily process: numeric representations. These representations are called **embeddings**.Sentence-BERT is a modification of the BERT architecture specifically designed to generate semantically meaningful sentence-level embeddings.

Dense Retrival & Semantic Search :

Dense Retrieval:

In the embedded vector space points that are close together mean that the text they represent is similar,this idea can be used to build search systems. When a search query is recieved, we embed the query, thus projecting it into the same space as the text archive. Then we simply find the nearest documents to the query in that space.

Semantic Search:

Semantic search enables searching by meaning, and not simple keyword matching.

The idea behind semantic search is to embed all entries in the corpus, whether they be sentences, paragraphs, or documents, into a vector space. During search, the query is embedded into the *same* vector space and the closest embeddings from your corpus are found. These entries should have a high **semantic similarity** with the query.

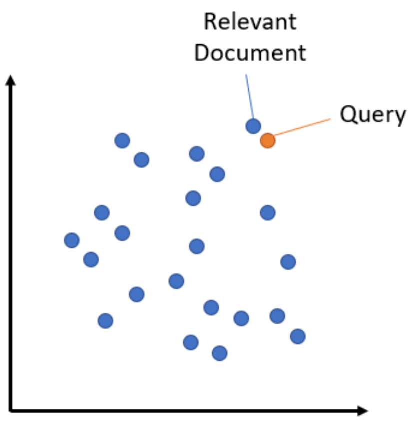


Figure 1: Dense Retrieval

FAISS :

As per the FAISS website

> ”Faiss is a library for efficient similarity search and clustering of dense vectors. It contains algorithms that search in sets of vectors of any size...It is developed primarily at FAIR, the fundamental AI research team of Meta.”

Given a set of vectors x_i in dimension d, **Faiss** builds a data structure (also known as an

index) in RAM from it. Once the index is constructed, given a new vector x in dimension d, it can efficiently perform nearest neighbor search.

$$j = \ argmin_i \|x - x_i\|$$

here $\|.\|$is the Euclidean distance L^2 and computing the argmin is the **search** operation on the index. We can also perform maximum inner product search , i.e . $argmax_i \ \langle x, x_i \rangle$,i.e. *Cosine Similarity* instead of Eucledian minimisation to obatin the nearest neighbour.

Quantization : We can LLMs efficient by reducing the memory requirements of the model's original weights before projecting them into smaller matrices. The weights of an LLM are numeric values with a given precision, which can be expressed by the number of bits like float64 or float32.

If we lower the amount of bits (4 or 8) to represent a value, we get a less accurate result. However, if we lower the number of bits we also lower the memory requirements of that model.

With quantization, we aim to lower the number of bits while still accurately representing the original weight values. when directly mapping higher precision values to lower precision values, multiple higher precision values might end up being represented by the same lower precision values.

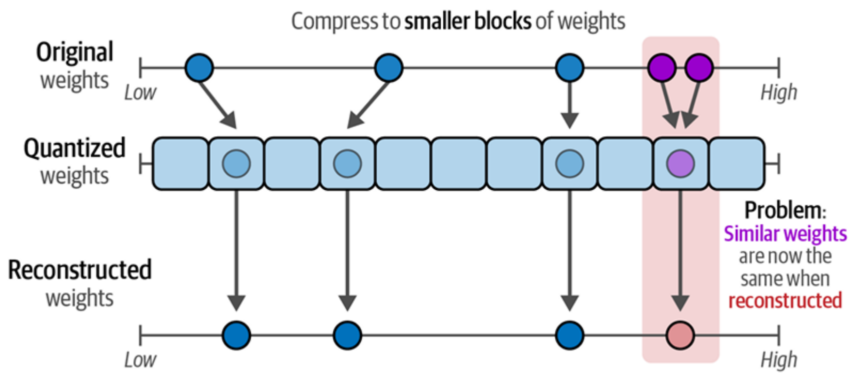

Compress to smaller blocks of weights
Original weights
Low
High
Quantized weights
Reconstructed weights
Low
High
Problem: Similar weights are now the same when reconstructed

Figure 2: Quantization

Reranking : A reranker takes in the query and a number of search results from the retriever, and returns the *optimal* ordering of these documents so the most relevant ones to the query are higher in ranking.

Vector-based retrieval enables efficient search by representing documents as dense embeddings. However, it can oversimplify the original content, especially when working with long documents that must be split into smaller chunks. This segmentation can lead to a loss of contextual continuity. Moreover, retrieval typically focuses only on the top-k most similar chunks, which risks overlooking other relevant segments. If the truly informative portions are not ranked highly by the vector similarity, they may be excluded from the final response. To address this, a re-ranking step is often introduced. This stage reassesses the initially retrieved candidates, aiming to surface the most contextually relevant and meaningful nodes for downstream tasks

Evaluation Modules:

- LlamaIndex has introduced **built-in evaluators** these include:
  - **FaithfulnessEvaluator:** It uses an LLM to judge whether responses are grounded in context.
  - **RelevancyEvaluator:** Evaluates alignment between the query and the generated response.
- **HuggingFace Evaluate:** An ecosystem of standard NLP evaluation metrics, such as BLEU (Papineni et al., 2002), ROUGE (Lin, 2004), and BERTScore (Zhang et al., 2019), consolidated under the `evaluate` library by HuggingFace. This open initiative standardized text generation evaluation across academia and industry.
- **RAGAS :** (Retrieval-Augmented Generation ASsessment) is an open-source framework designed to evaluate the performance of RAG systems. Traditional evaluation metrics often fall short in capturing the nuanced quality of RAG outputs, especially in terms of factual accuracy and relevance. RAGAS provides a set of *automated and interpretable evaluation metrics* thus making it suitable for benchmarking, debugging, and improving RAG-based applications.

  The framework assesses generated responses using multiple dimensions , we will be utilizing the following metrics:
  - **Context Precision**: Checks how much of the retrieved context is actually relevant to the answer.
  - **Context Recall**: Evaluates whether all necessary information from the context was used.

2 Methodology

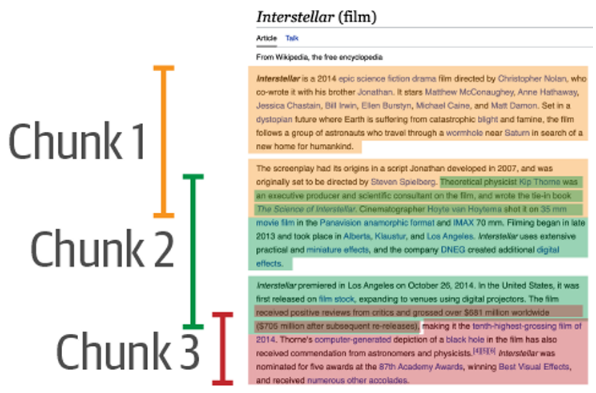
1. **Document Ingestion**:

1. Text files were **loaded** using `SimpleDirectoryReader` in a document object . It will select the best file reader based on the file extensions.There is provision for custom file extractors as well.

2. Transformer language models are limited in context sizes, very long texts that go above the number of words or tokens that the model support can not be fed directly,thus the ingested text is thus split into **chunks** , here we use `TextTokenSplitter` for this .

   Many chunks derive a lot of their meaning from the text around them. We incorporate some context by adding some of the text before and after them . This way, the chunks can overlap so they include some surrounding text that also appears in adjacent chunks.

   We are using a `chunk_overlap` of 200 , `Chunk Size` of 1024



Interstellar (film)

From Wikipedia, the free encyclopedia

Interstellar is a 2014 epic science fiction drama film directed by Christopher Nolan, who co-wrote it with his brother Jonathan. It stars Matthew McConaughey, Anne Hathaway, Jessica Chastain, Bill Irwin, Ellen Burstyn, Michael Caine, and Matt Damon. Set in a dystopian future where Earth is suffering from catastrophic blight and famine, the film follows a group of astronauts who travel through a wormhole near Saturn in search of a new home for humankind.

The screenplay had its origins in a script Jonathan developed in 2007, and was originally set to be directed by Steven Spielberg. Theoretical physicist Kip Thorne was an executive producer and scientific consultant on the film, and wrote the tie-in book *The Science of Interstellar*. Cinematographer Hoyte van Hoytema shot it on 35 mm movie film in the Panavision anamorphic format and IMAX 70 mm. Filming began in late 2013 and took place in Alberta, Klaustur, and Los Angeles. *Interstellar* uses extensive practical and miniature effects, and the company DNEG created additional digital effects.

Interstellar premiered in Los Angeles on October 26, 2014. In the United States, it was first released on film stock, expanding to venues using digital projectors. The film received positive reviews from critics and grossed over $681 million worldwide ($705 million after subsequent re-releases), making it the tenth-highest-grossing film of 2014. Thorne's computer-generated depiction of a black hole in the film has also received commendation from astronomers and physicists.[4][5][6] *Interstellar* was nominated for five awards at the 87th Academy Awards, winning Best Visual Effects, and received numerous other accolades.

Figure 3: Chunking Overlapping

2. **Embedding and Indexing**: Embeddings turn text into numeric representations.Chunks generated from the previos step are converted into embeddings using sentence embedding model SBERT:`all-MiniLM-L6-v2`. It maps extracted sentences & paragraphs to a **384** dimensional dense vector space that can be used for semantic search.The embeddings generated are stored in a FAISS-based `VectorStoreIndex`.

3. **Retrieval**:

   For each query, relevant documents were retrieved using vector similarity search. using the `VectorIndexRetriever` method with `similaritytopK` set to 5. Using `similaritytopK` we specify that the retriever should fetch the top 5 nodes that match the query.

   `RetrieverQueryEngine` was utilised to enforce the query engine that fetches our response nodes a.k.a **context**, there is also a provision for node postprocessing.

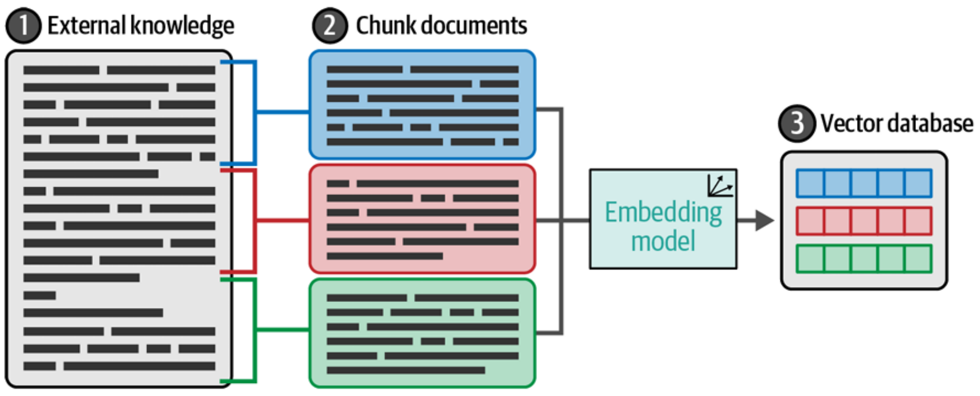


Figure 4: Document Ingestion and Indexing

4. **Reranking :** We are using the `LLMRerank` node post processing module. The choice of batch size is 5 . The topN value is 5.

   - A query string is generated for each batch of nodes.
   - The batch is formatted using `QueryBundle` and passed to the language model along with the query and a prompt that specifies how to choose relevant items.
   - The output from the language model is parsed to extract possible choices and their corresponding relevance scores.
   - The extracted choices are sorted in descending order based on relevance.
   - The top n most relevant results are selected and returned.

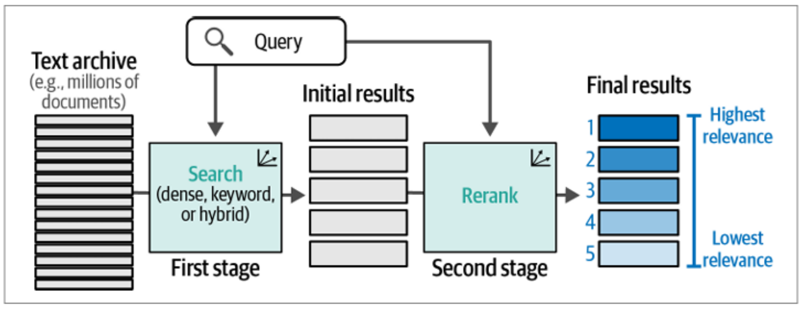


Figure 5: Rerank

5. **Response Generation**: `Qwen3-1.7B` was used to generate answers using a prompt and retrieved context. The key parameters were as follows :

   - `Context_Window` : `1024`
   - `Max_New_Tokens` : `512`
   - `Temeperature` : different values
   - `Topk` :5

- `System_prompt` : ”You are a Q&A assistant. Your goal is to answer questions as accurately as possible based on the instructions and context provided.If it is not in the context, say you don’t know. Don’t try to make up an answer. no extra words”

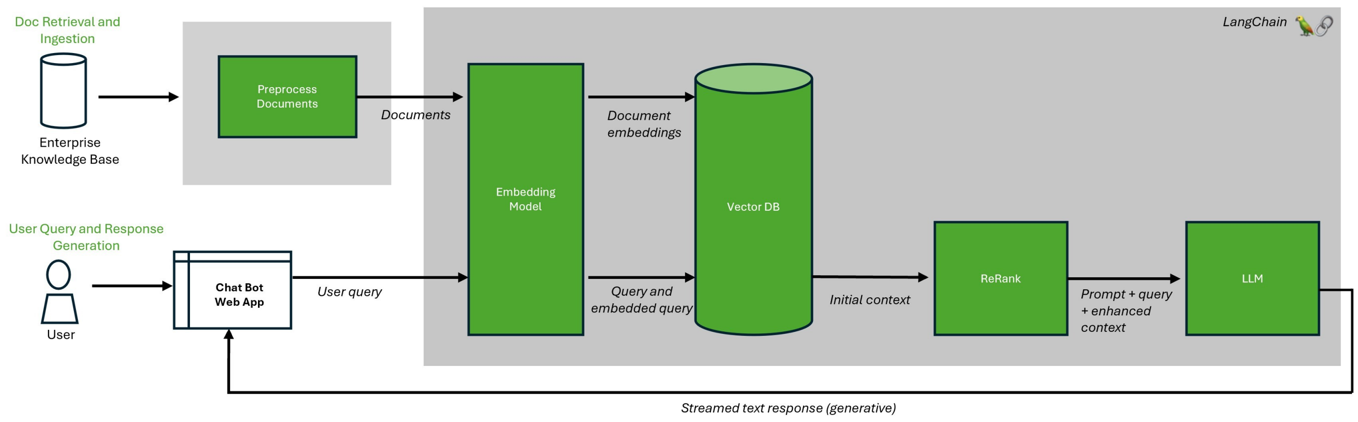


Figure 6: RAG Pipeline with Rerank *Source*

6. **Evaluation**: Responses were scored using BLEU, ROUGE, BERTScore, and LlamaIndex’s `FaithfulnessEvaluator` and `RelevancyEvaluator`.

   Also `LLMContextRecall` and `LLMContextPrecisionWithoutReference` were used from the RAGAS library

3 Evaluation Metrics

The RAG was trained on Personal finance and Fundamental Analysis modules from Varsity by Zerodha.

3.1 Evaluation by Character Overlap :

n-grams:[1] An n-gram is a sequence of n words: a 2-gram (which we’ll call **bigram**) is a two-word sequence of words like The water, or water of, and a 3- gram (**trigram**) is a three-word sequence of words like `The water of, or water of Walden`.

[1]**https://web.stanford.edu/~jurafsky/slp3. Chapter 3**.

3.1.1 BLEU : The Bilingual Evaluation Understudy

The BLEU score for a corpus of candidate translation sentences is a function of the n-gram word precision over all the sentences combined with a brevity penalty computed over the corpus as a whole.

n-garm precision: We evaluate precision by calculating the count of n-grams (combination of n words), in the generated text appearing in the reference or **”Ground Truth”** answer. The unigram precision for this corpus is the percentage of unigram tokens in the candidate translation that also occur in the reference translation, and ditto for bigrams, and so on, up to 4-grams.[2]

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

Where:

- p_n is the modified n-gram precision.
- w_n is the weight for each n-gram size.
- BP is the brevity penalty: BP = 1 if c is more than r and $\exp(1 - \frac{r}{c}) otherwise$

where c is the length of the candidate sentence, and r is the length of the reference sentence

3.1.2 ROUGE Score

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics for evaluating automatic summarization and machine translation. It measures the overlap between n-grams, word sequences, and word pairs in the generated (candidate) and reference summaries.

ROUGE is especially recall-focused, making it suitable for tasks like **summarization**, where the generated text is typically shorter than the reference.

Key ROUGE Variants :

- **ROUGE-N:** Measures the n-gram overlap between the candidate and reference.

[2]**https://web.stanford.edu/~jurafsky/slp3. Chapter 13** .

$$ROUGE - N = \frac{Number\ of\ matching\ n-grams}{Total\ n-grams\ in\ reference}$$

- **ROUGE-L:** Based on the Longest Common Subsequence (LCS), capturing sentence-level structure similarity.

$$ROUGE - L = F_\beta = \frac{(1+\beta^2)\cdot P \cdot R}{\beta^2 \cdot P + R}$$

  where:

  - P = precision
  - R = recall
  - β is typically set to 1

3.2 Automatic Evaluation: Embedding-Based Methods

BERTScore: A semantic similarity metric based on contextual embeddings from pre-trained models like BERT and RoBERTa. It correlates well with human judgment.

We can measure the similarity of reference x and generated response $\tilde{x}$ by the similarity of their embeddings. The BERTScore algorithm,[3] passes the reference x and the candidate $\tilde{x}$ through BERT, computing a BERT embedding for each token x_i and $\tilde{x}_j$. Each pair of tokens $(x_i, \tilde{x}_j)$ is scored by its cosine similarity:

$$\frac{x_i \cdot \tilde{x}_j}{\|x_i\|\|\tilde{x}_j\|}$$

Each token in x is matched to a token in $\tilde{x}$ to compute recall, and each token in $\tilde{x}$ is matched to a token in x to compute precision (with each token greedily matched to the most similar token in the corresponding sentence). BERTScore provides precision and recall .F_1 is the harmonic mean of precision and recall.

$$R_{BERT} = \frac{1}{|x|}\sum_{x_i \in x} \max_{\tilde{x}_j \in \tilde{x}} x_i \cdot \tilde{x}_j \qquad P_{BERT} = \frac{1}{|\tilde{x}|}\sum_{\tilde{x}_j \in \tilde{x}} \max_{x_i \in x} x_i \cdot \tilde{x}_j$$

[3]**arXiv:1904.09675**.

3.3 RAG Specific metrics for Hallucination:[4]

3.3.1 Faithfulness:

The Faithfulness metric measures how factually consistent a response is with the retrieved context. It ranges from 0 to 1, with higher scores indicating better consistency.

A response is considered faithful if all its claims can be supported by the retrieved context.

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

We are using the in-built `FaithfulnessEvaluator` for our evaluation . We use `llama370b` as a judge LLM for evaluation via a groqAPI

3.3.2 Relevancy:

Measures how relevant a response is to the user input. Higher scores indicate better alignment with the user input, while lower scores are given if the response is incomplete or includes redundant information.

1. A set of artificial questions are generated based on the response. These questions are designed to reflect the content of the response.
2. We compute the cosine similarity between the embedding of the user input i.e the **prompt** and the embedding of each generated question
3. Average of these cosine similarity scores gives us **Relevancy**

We are using the in-built `RelevancyEvaluator` for our evaluation . We use `llama370b` as a judge LLM for evaluation via a groqAPI.

3.3.3 Context Precision:[5]

Context Precision is a metric that measures the proportion of relevant chunks in the retrieved contexts. It is calculated as the mean of the **precision@k** for each chunk in the context.

[4]**https://docs.ragas.io/en/stable/concepts/metrics/available˙metrics/**.
[5]**https://docs.ragas.io/en/stable/concepts/metrics/available˙metrics/context˙precision/**.

Precision@k is the ratio of the number of relevant chunks at rank k to the total number of chunks at rank k

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

$$Context\ Precision@K = \frac{\sum_{k=1}^{K}(Precision@k \times v_k)}{Total\ numberof\ relevant\ items\ in\ the\ topKresults}$$

Here K is the number of chunks in the retrieved contexts and v_k is a relevance indicator at rank k.

We use `LLMContextPrecisionWithoutReference` from RAGAS for evaluation, using `deepseek-r1-distill-llama-70b`[6] for comparison of each chunk present in retrieved contexts with response

3.3.4 Context Recall:[7]

Context Recall measures how many of the relevant documents (or pieces of information) were successfully retrieved. It focuses on not missing important results. Higher recall means fewer relevant documents were left out.

To estimate context recall from the reference a.k.a the Ground Truth answer, the reference is broken down into claims each claim in the reference answer is analyzed to determine whether it can be attributed to the retrieved context or not. In an ideal scenario, all claims in the reference answer should be attributable to the retrieved context. This evaluation is performed using a LLM.

We use `LLMContextRecall` from RAGAS for evaluation, using `deepseek-r1-distill-llama-70b`.

4 Results

Table 1: Evaluation Metrics Across Different Temperature Settings

Temp	**BLEU**	**ROUGE-1**	**ROUGE-2**	**BERTScore F1**	**Faithful**	**Relevant**	**C-Recall**	**C-Precision**
0.2	0.0299	0.3469	0.0730	0.8200	0.0	1.0	0.429	0.450
0.4	0.0220	0.3467	0.1238	0.8241	1.0	1.0	0.536	0.806
0.5	0.0270	0.3713	0.0781	0.8147	0.0	1.0	0.333	0.200
0.7	0.0790	0.3248	0.1465	0.8234	1.0	1.0	–	–

[6] **https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Llama-70B**.

[7] **https://docs.ragas.io/en/stable/concepts/metrics/available˙metrics/context˙recall/**.

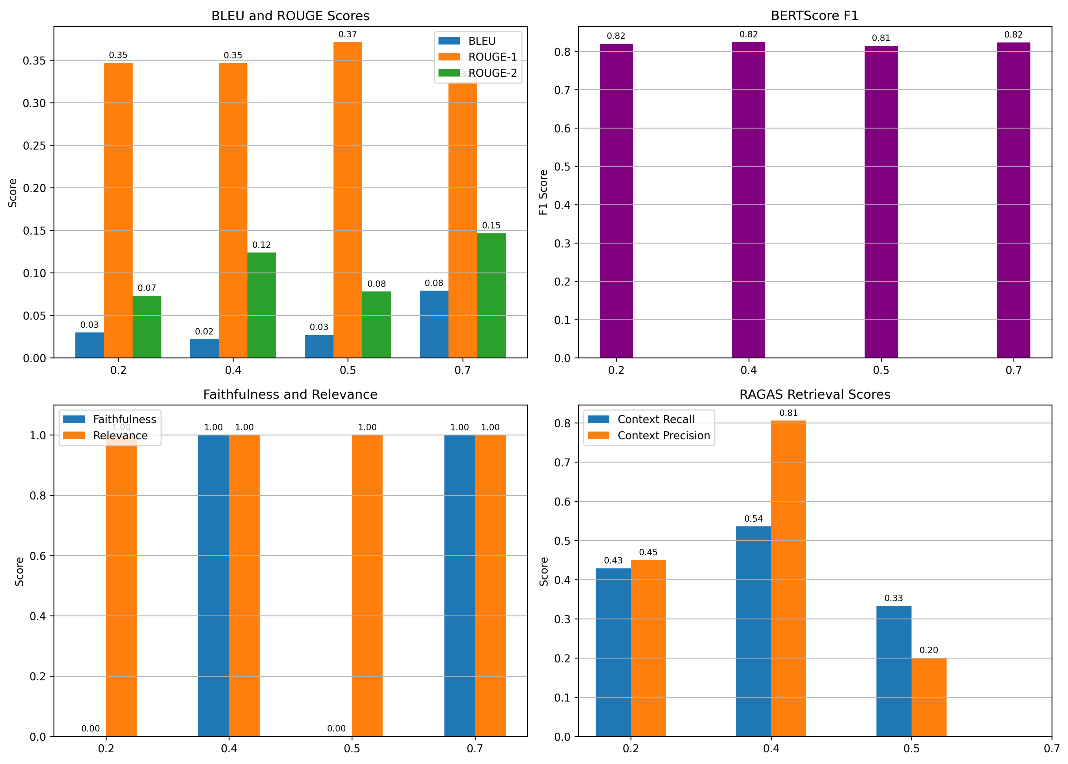


Figure 7: Evaluation Overview

5 Evaluation Analysis and Interpretation

This section provides a detailed analysis of evaluation metrics across different temperature settings (0.2, 0.4, 0.5, 0.7). The metrics analyzed include BLEU, ROUGE, BERTScore, Faithfulness, Relevance, and RAGAS Retrieval scores.

5.1 BLEU and ROUGE Scores

- BLEU score increases significantly at temperature 0.7, indicating better n-gram overlap due to more diverse outputs.
- ROUGE-2 also improves with higher temperature, peaking at 0.7, suggesting enhanced phrase-level fluency.
- ROUGE-1 remains relatively stable, showing consistent word-level matching across all temperatures.

5.2 BERTScore (F1)

- BERTScore remains high and stable across all temperatures, ranging from 0.815 to 0.824.
- Indicates strong semantic similarity between generated and reference text, regardless of temperature setting.

5.3 Faithfulness and Relevance (LLM Judgments)

- Faithfulness is perfect (1.0) at temperatures 0.4 and 0.7 but drops to 0.0 at 0.2 and 0.5.
- Relevance remains perfect (1.0) for all temperatures, showing that generated answers stay on-topic.
- Temperature 0.4 and 0.7 produce the most trustworthy and grounded outputs.

5.4 RAGAS Retrieval Metrics

- Context Precision is highest at temperature 0.4 (0.806), indicating high quality of retrieved documents.
- Context Recall is also highest at 0.4 (0.536), suggesting better coverage of required information.
- Temperature 0.5 shows the poorest retrieval performance.

6 Conclusion

The Qwen model demonstrates strong generative capability within a RAG setup. The use of LlamaIndex enables precise retrieval, while the integration of semantic and lexical evaluation metrics provides a multidimensional perspective on output quality.

7 Future Directions

- Benchmark Qwen3-1.7B against Mistral, Claude, and GPT-4 in the same pipeline.

- Deploy this on Steamlit or Docker, Heroku etc.

8 References:

- Jay Alammar, Maarten Grootendorst. 2024. Hands-On Large Language Models
- Daniel Jurafsky and James H. Martin. 2025. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition.
- RAGAS Documentation
- LlamaIndex Documentation
- Varsity by Zerodha
- Google and ChatGPT