STAR WARS: BROTHERHOOD RAG

Table of Contents

Introduction

This document outlines the development of a Retrieval-Augmented Generation (RAG) based question-answering system designed to answer natural language queries about the novel *Star Wars: Brotherhood*. The system addresses the challenge of efficiently retrieving and generating relevant information from a specific domain corpus—in this case, a single PDF of the novel. By leveraging advanced retrieval and generation techniques, the RAG system ensures accurate and context-aware responses. The use case for this RAG system lies in providing an interactive tool for readers, researchers, or fans who seek detailed insights or quick answers related to the novel's content, thereby enhancing their engagement and understanding of the material.

Platform Details

For all stages of the project, including PDF parsing, data cleaning, experimentation, and model execution, the Kaggle platform was utilized. Kaggle's environment provided the necessary computational resources, such as GPUs, which facilitated efficient processing and experimentation. All processes were conducted within Kaggle notebooks, ensuring consistency and ease of resource management. No other platforms were used during the development and testing of the system.

Data Details

The data used for this project consisted of a single PDF document of the book ***Star Wars: Brotherhood*.** The source of the document was LibGen, a repository containing thousands of PDFs of various books. The PDF is approximately 352 pages long and served as the sole corpus for the system. The pdf was parsed through the pdfPlumber module.

```python
import pdfplumber

# Open the PDF file
with pdfplumber.open('/kaggle/input/star-wars/Star Wars - Brotherhood Mike Chen.pdf') as pdf:
    # Open the text file for writing
    with open('knowledge_base.txt', 'w', encoding='utf-8') as output:
        for page in pdf.pages:
            text = page.extract_text() or ""  # Handle cases where text is None
            output.write(text + '\n')  # Write text to file with a newline
```

Figure 1: Code snippet for parsing pdf (pdfplumber)

Algorithms, Models, and Retrieval Methods

Retrieval Methods

The retrieval phase of the system utilized a hybrid approach, combining the strengths of both semantic search and keyword-based search using dense embeddings retriever and BM-25 techniques to ensure high-quality document retrieval.

1. **BM25 Function:** BM25, a probabilistic retrieval model, was used to rank documents based on the frequency of query terms adjusted by term saturation and document length normalization. It excelled at capturing keyword-based relevance, making it effective for queries requiring exact term matches.
2. **Dense Embeddings Retriever:** Dense retrieval was implemented using pre-trained transformer-based models to generate semantic embeddings (BAAI/bge-small-en, GPT4AllEmbedgins). This approach allowed for understanding the contextual relationships between queries and documents, improving retrieval for natural language queries.
3. **Reciprocal Rank Fusion (RRF):** The results from BM25 and the dense embedding retriever were combined using RRF. This method prioritized documents ranked highly by both retrieval strategies, selecting the top n documents from the fused results. This hybrid mechanism ensured precision and semantic relevance in retrieval.

Justification for the Approach

Combining BM25 with dense embeddings retrieval provided a balance between precision and recall. BM25 addressed keyword-matching scenarios, while dense embeddings captured semantic nuances. RRF integration leveraged the strengths of both methods, resulting in a robust and versatile retrieval system.

Standalone BM25 performed well for exact term queries but struggled with semantic comprehension. Dense retrieval captured context effectively but occasionally prioritized less relevant documents due to limited reliance on keyword matches. The hybrid approach with RRF resolved these limitations, delivering improved overall performance for diverse query types.

Models

For the generation phase of the system, multiple state-of-the-art large language models (LLMs) were employed to evaluate their effectiveness in answering questions based on the retrieved documents. The specific models used were

- **Falcon-3B-Instruct**
- **Falcon-7B-Instruct**
- **Qwen/Qwen2.5-3B-Instruct**
- **Qwen/Qwen2.5-7B-Instruct**

Explanation for Choices

- **Falcon Models:** Chosen for their efficient architecture and strong performance in generating coherent and contextually relevant responses. The instruct fine-tuned versions enhanced alignment with user queries.
- **Qwen Models:** Known for their structured understanding and contextual reasoning. The 2.5 versions provided improved capabilities over earlier iterations.

Using both 3B and 7B variants of Falcon, Qwen, and Mistral models ensured that the system could cater to a broad range of scenarios, from lightweight tasks to those requiring extensive reasoning. This approach facilitated a comprehensive evaluation and optimization of the system for diverse user requirements.

Employing multiple LLMs allowed for comparative analysis across various parameters, such as:

- **Performance:** To identify the best-performing model for different query types.
- **Efficiency:** To evaluate the trade-offs between latency and answer quality.

Methodology for Implementing RAG

In this section, we will explain how we implemented the RAG model, using three main stages:

1. Vector DB Setup Stage
2. Retrieval Stage
3. Generation Stage

By following these three stages, we were able to implement the RAG model, which can accurately and efficiently answer readers' queries using the books' pdf as the knowledge source. In the following sections, we will provide more details on each stage and discuss the results of our implementation.

During the development of the RAG model, we conducted various tests to evaluate the performance of the system and identify areas for improvement. Based on the results of these tests, we made several iterations to the model to enhance its accuracy and efficiency. Overall, the RAG model was improved over time through a series of tests, iterations, and evaluations, resulting in highly accurate and efficient responses.

Vector DB Setup Stage

1. **RAG Read Docs From the Source:**

   The RAG system begins the process by reading documents from the data source. In our case, we used the books' PDF as the data source mentioned earlier, and we parsed the pdf using pdfPlumber and removed the starting and the ending pages of the pdf since they don't make much sense that were the appendix, references and other pages like these. We then concatenated all this data into a single text file.

2. **Send Docs to Chunk Function:**

   The RAG system sends the documents to a chunking function. This step involves breaking down the documents into smaller pieces which are called "chunks". We experimented with various chunking techniques, including character chunking, recursive splitting, and semantic splitting.

   - Character Chunking: We tested different separators, chunk sizes (200, 600, 1000), and overlaps (100, 200).
   - Recursive Split: We compared the performance of the recursive method with that of the character text splitter using the same chunk size and overlap. The only difference between the two methods was the list of separators used. Surprisingly, the character text splitter outperformed the recursive method in this comparison.

3. **Send Chunks to Embedding Function:**

   The RAG pipeline forwards the document chunks to an embedding function for further processing. In our experimentation, we used *GPT4AllEmbeddings, BAAI/bge-small-en*. The primary function of the embedding process is to convert the chunks into numerical vectors, known as embeddings, that encapsulate the semantic meaning of the text.

4. **Store Vectors in DB:**

   Finally, these vectors are stored in a vector database. The vector database used by us was *FAISS*.

```python
# Initialize embedding model
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")

# Create FAISS vector database
vectordb = FAISS.from_documents(chunks, embedding_model)

# Save FAISS index to disk for later use
vectordb.save_local("faiss_index")

# Check the number of stored documents
print(f"Number of documents in the vector store: {vectordb.index.ntotal}")
```

Figure 2: Code snippet for saving embeddings to vector DB

Retrieval Stage

1. **User Submits Query:**

   The process begins with the user submitting a search query to the system.

2. **Search Similar Vectors in DB:**

   The RAG system processes the user's query by converting it into an embedding vector using the same embedding function used before. This vector is then used to search for the most similar vectors in the document database (Vector DB) which in our case is *FAISS*. The system retrieves the top k results with the highest similarity scores to the user's query, where k is a configurable parameter. We experimented with different values of k like *(5,10,15,20)* and found that a value of 15 was optimal for our use case.

   Several retrieval techniques were used for experimentation:

- **BM-25:**

We experimented with a BM25 retriever. Unlike the vector-based approach, BM25 does not rely on embeddings and instead searches for keywords within the documents. For our test data, the BM25 retriever performed average, but performed very accurately particularly when the queries were specific and contained unique keywords or phrases. In fact, the BM25 retriever often outperformed the vector-based retrieval method in terms of accuracy and relevance. This can be attributed to the fact that BM25 is specifically designed to handle keyword-based searches and assigns higher relevance scores to documents that contain the exact query terms. As a result, it is able to quickly and efficiently identify the most relevant documents, even in large and complex datasets.

```
for doc, score in results_embedding:
    print(f"page {doc.metadata.get('page','Unknown')} - Score: {score:.4f} - {doc.page_content[:100]}...")

# Get BM25 scores for all documents and sort to get top-k results
results_bm25 = [(idx, bm25.get_scores(query.split())[idx]) for idx in range(len(texts))]
results_bm25 = sorted(results_bm25, key=lambda x: x[1], reverse=True)[:k]  # Keep only top-k results
# Convert BM25 results to (Document, score) format
results_bm25_docs = [(Document(page_content=texts[idx], metadata=metadata[idx]), score) for idx, score in results_bm25]

print("************BM25 Results*************")
for doc, score in results_bm25_docs:
    print(f"page {doc.metadata.get('page','Unknown')} - Score: {score:.4f} - {doc.page_content[:100]}...")
```

Figure 3: Code snippet for BM-25 retriever

- **Dense Retriever:**

We experimented with a Dense Retriever, which uses embeddings to capture semantic similarities between queries and documents. Unlike BM25, it excels with less specific queries by identifying contextually relevant matches even without exact keyword overlap. On our test data, the Dense Retriever performed well, often retrieving documents missed by BM25, thanks to its ability to understand nuanced meanings. This makes it a powerful tool for retrieving relevant information in complex and varied datasets.

```
query_embedding = embedding_model.embed_query(query)
results_embedding = vectordb.similarity_search_with_score_by_vector(query_embedding, k=k)
results_embedding = sorted(results_embedding, key=lambda x: x[1], reverse=True)

print("===========Dense Embeddings============")
for doc, score in results_embedding:
    print(f"page {doc.metadata.get('page','Unknown')} - Score: {score:.4f} - {doc.page_content[:100]}...")
```

Figure 4: Code snippet for Dense retriever

- **Hybrid Approach (Reciprocal Rank Function):**

We explored RRF as a method to combine results from multiple retrieval strategies in Retrieval-Augmented Generation (RAG). By prioritizing documents consistently ranked highly across methods, RRF enhanced the relevance and quality of the retrieved information, proving effective in improving the input for generative models.

```
def reciprocal_rank_fusion(results_bm25, results_embedding, k=2):
    scores = {}

    # Use document content or metadata as the key
    for rank, (doc, score) in enumerate(results_bm25):
        doc_id = doc.page_content  # Or use doc.metadata.get("source", "unknown") if available
        scores[doc_id] = scores.get(doc_id, 0) + 1 / (rank+1) # (k + rank + 1)
        print("BM25", scores[doc_id])

    for rank, (doc, score) in enumerate(results_embedding):
        doc_id = doc.page_content  # Use the same identifier
        scores[doc_id] = scores.get(doc_id, 0) + 1 / (rank+1) # (k + rank + 1)
        print("Dense", scores[doc_id])

    return sorted(scores.items(), key=lambda x: x[1], reverse=True)
```

Figure 5: Code snippet for Reciprocal Rank Function

Generation Stage

The RAG system receives the reranked documents from the retrieval stage and limits them to the top k documents. In our experiments, we used k *(3,5,10,15)* to test on different parameters.

Long-Context reordering:

The performance of our model is negatively impacted when it is presented with more than five long documents, each containing 1000 characters. This is because models tend to disregard the provided documents when attempting to extract relevant information from lengthy contexts. To avoid this issue, we reordered the documents after retrieval to improve the model's performance.

```
def reorder_sorted_responses(sorted_responses):
    # Alternate between most important (edges) and least important (center)
    most_important = sorted_responses[::2]  # Take every other response starting with the first
    least_important = sorted_responses[1::2]  # Take every other response starting with the second

    # Merge: Place least important in the center
    reordered_responses = []
    while most_important or least_important:
        if most_important:
            reordered_responses.append(most_important.pop(0))  # Add from most important
        if least_important:
            reordered_responses.append(least_important.pop())  # Add from least important

    return reordered_responses
reordered_responses = reorder_sorted_responses(retrieved_responses)
```

Figure 6: Code snippet for Long Context Reordering

Summarization:

The performance of our model did not improve when summarization was applied to the retrieved context. Instead, the model generated incorrect answers, as it lacked the detailed knowledge required to infer and construct accurate responses. Since we are developing a RAG system for a novel, answering specific queries demands access to comprehensive and detailed context. Summarization reduces the richness of the information, which is critical for the model to understand and provide accurate statements. Therefore, retaining detailed context is essential for achieving the desired accuracy and reliability in our system.

```
def generate_lsa_summary(retrieved_responses, num_summary_sentence=50):
    # Combine the retrieved responses into one string
    text = " ".join(retrieved_responses)

    # Initialize LSA summarizer
    LANGUAGE = "english"
    parser = PlaintextParser.from_string(text, Tokenizer(LANGUAGE))
    lsa_summarizer = LsaSummarizer()

    # Generate the summary
    summary = []
    for sentence in lsa_summarizer(parser.document, num_summary_sentence):
        summary.append(str(sentence))

    # Join the summarized sentences and wrap them for better readability
    summarized_text = " ".join(summary)
    return textwrap.fill(summarized_text, 100)
```

Figure 7: Code snippet for Summarization

Prompt:

The RAG system then combines the original user query with the top k documents, using a specific format to ensure that the LLM can process the information correctly.

Finally, the combined query and documents are sent to a large language model (LLM), which generates an answer based on the information provided.

```
# ### **Summarized Retrieved Information**:
# {summarized_responses}

# Construct the RAG prompt
prompt = f"""
You are an AI assistant tasked with answering questions based on retrieved knowledge.

# ### **Retrieved Information**:
# 1. {reordered_responses[0]}
# 2. {reordered_responses[1]}
# 3. {reordered_responses[2]}
# 4. {reordered_responses[3]}
# 5. {reordered_responses[4]}

### **Question**:
{question}

### **Instructions**:
- Integrate the key points from all retrieved responses into a **cohesive, well-structured answer**.
- If the responses are **contradictory**, mention the different perspectives.
- If none of the retrieved responses contain relevant information, reply:
  **"I couldn't find a good response to your query in the database."**
"""
```

Figure 8: Prompt to LLM

LLM Used:

We experimented using different LLMs to find the LLM which gives the most relevant and accurate answer to the users query. We cannot use a LLM with very large parameters since Kaggle has limited GPU memory so we were limited to 7B parameters. Secondly if we use a LLM with very large parameters it was a potential risk that the LLM would know about the novel as that novel would be included in its knowledge base but the LLMs which we tested our on all of them didn't know anything about this novel.

Performance Metrics

Test Queries

To evaluate the performance of our RAG, we tested it with a set of questions that readers commonly ask. These questions included:

- What is the name of the Neimoidian guard who assists Obi-Wan?
- What was the cause of the bombing on Cato Neimoidia in Brotherhood?
- What is the name of the Jedi Padawan accompanying Obi-Wan in this mission?
- What tragedy prompts Obi-Wan to travel to Cato Neimoidia?
- What does Anakin Skywalker gifts Padmé Amidala during the night at the Uhmandasee Market and Please Explain its design?

Embedding Used	Chunk Size	Chunk Overlap	Vector DB	BM-25	Dense Retriever	RRF	LLM used	Summary	Reorder	Correct	Time taken
BAAI/bge-small-en	200	50	FAISS	✓	✓	✓	tiiuae/Falcon3-3B-Instruct	×	×	2/5	52.3s
BAAI/bge-small-en	512	200	FAISS	✓	✓	✓	Qwen/Qwen2.5-3B-Instruct	×	×	2/5	18.78s
BAAI/bge-small-en	512	200	FAISS	✓	×	×	Qwen/Qwen2.5-3B-Instruct	×	×	1/5	20.36s
BAAI/bge-small-en	400	100	FAISS	✓ k=15	×	×	tiiuae/Falcon3-7B-Instruct	×	×	3/5	40.57s
BAAI/bge-small-en	1000	512	FAISS	×	✓	×	Qwen/Qwen2.5-7B-Instruct	×	×	3/5	22.4s
BAAI/bge-small-en	1000	512	FAISS	✓	✓	✓	Qwen/Qwen2.5-7B-Instruct	×	×	2/5	45.4s
BAAI/bge-small-en	1000	512	FAISS	×	✓	×	Qwen/Qwen2.5-7B-Instruct	✓	×	3/5	32.0s
BAAI/bge-small-en	1000	256	FAISS	×	✓	×	Qwen/Qwen2.5-7B-Instruct	×	✓	4/5	26.2s
sentence-transformers/all-MiniLM-L6-v2	1000	256	FAISS	×	✓	×	Qwen/Qwen2.5-7B-Instruct	×	✓	2/5	15.2s
BAAI/bge-small-en	1200	600	FAISS	×	✓ k=6	×	Qwen/Qwen2.5-7B-Instruct	×	×	5/5	35.67s

Figure 9: Tests

Chunk Sizes

We can see from figure 9 that when smaller chunk sizes (e.g., 200–400 tokens) were used in the RAG system, the accuracy of answers was notably lower, with only 1/5 or 2/5 correct responses in most test cases. This suggests that fine-grained chunks may fragment contextual information, leading to incomplete or irrelevant retrievals. Additionally, smaller chunks often required longer processing times (e.g., 40.57s for 400-token chunks vs. 26.2s for 1000-token chunks), likely due to the increased number of retrievals.

Models

The **Qwen2.5-3B-Instruct** model, being relatively smaller with only 3 billion parameters, tends to prioritize surface-level connections between words rather than deeply analyzing multi-sentence relationships.

```
Based on the information provided, Anakin Skywalker gifts Padmé Amidala a piece
of his Padawan braid encased in carbonite, which is contained within a simple
gold pendant with a metallic spiral design.  The pendant symbolizes "new
```

In this case, while it correctly recognized that Anakin gifted Padmé a pendant involving his Padawan braid, it misunderstood the fine detail regarding the pendant's design. It mistakenly implied that the pendant itself had a metallic spiral design, when in fact, it was the coiled braid inside the pendant that formed the spiral.

The **Qwen2.5-7B-Instruct** model, with its larger 7 billion parameter size, demonstrated a much stronger ability to understand complex object relationships and metaphorical meaning. It correctly interpreted that the pendant held Anakin's Padawan braid, which was coiled into a spiral and encased within the pendant's simple structure .

```
his Padawan braid, which has been coiled into a spiral and encased in carbonite.
```

Retrieval Methods

BM25 Behavior (Qwen 3B):

The BM25 retriever struggled to find a precise or complete answer to the question about the cause of the bombing on Cato Neimoidia in *Brotherhood*. BM25 relies on exact or near-exact keyword matching between the query and the indexed text chunks, without truly "understanding" the meaning behind words. Because the provided passages didn't use explicit language like "the cause of the bombing was...", BM25 couldn't align the keywords from the query with a satisfying answer. As a result, the model correctly acknowledged that it couldn't find a definitive answer, but it also missed the deeper thematic clues spread across the text that hinted at political motivations and extremist actions.

```
I couldn't find a good response to your query in the database. The provided
passages do not explicitly state the cause of the bombing on Cato Neimoidia.
```

BM25's weakness here stems from its shallow retrieval mechanism: it focuses purely on lexical overlap rather than conceptual similarity. In books like *Brotherhood*, where the story often weaves the cause of events through multiple characters' perspectives and layers of subtlety, simple keyword-based methods are prone to miss the bigger picture.

Dense Retriever Behavior (Qwen 3B):

The dense retriever performed significantly better because it understands **semantic meaning** rather than relying on keyword overlap. Dense retrieval uses embedding-based models to capture the conceptual similarity between the question and the document chunks. This allowed the retriever to gather and synthesize multiple pieces of information: Obi-Wan attributing the bombing to extremism, Yoda and Ki-Adi-Mundi connecting it to Neimoidian grievances and political factors, and the general atmosphere of war and fear fueling the event.

```
Based on the information provided, there are conflicting perspectives regarding
the cause of the bombing on Cato Neimoidia.From Obi-Wan Kenobi's point of view,
the bombing appears to be an act of extremism by an extremist group,
unrelated to the broader conflict between the Republic and the Trade Federation.
```

Thanks to this semantic ability, the dense retriever recognized that the cause wasn't a single clear-cut reason but a combination of extremist ideology, wartime unrest, and political dissatisfaction. It even correctly highlighted that there are conflicting perspectives and that the exact cause remains somewhat ambiguous — which mirrors the storytelling style of *Brotherhood*. Dense retrieval's strength here lies in its ability to reason across scattered information, piece together underlying meanings, and present a more nuanced, multi-faceted answer even when the text doesn't handhold the reader.

Evaluation Stage

To comprehensively evaluate the performance of our RAG system, we employed two distinct methods: **LLM as a Judge Evaluation** and **Automated Evaluation using RAGAS**. These methods provided a multifaceted analysis of the system's performance across various metrics.

LLM as a Judge Evaluation

In this approach, we used ChatGPT to both generate questions and evaluate the RAG-generated responses. The steps were as follows:

1. **Question Generation**: ChatGPT generated domain-specific questions to test the system's ability to retrieve and utilize relevant contextual information.

2. **Answer Generation**: These questions were passed to the RAG system, which combined the user query with retrieved documents to generate answers.

3. **Answer Evaluation**: The RAG-generated answers were evaluated by ChatGPT based on a comparison with the expected answers. Scores were assigned based on the following parameters:

   - **Faithfulness**: Accuracy of the response in reflecting the retrieved context.
   - **Relevance**: Alignment of the answer with the question asked.
   - **Accuracy**: Overall Accuracy of the answer.

This method provided valuable qualitative and quantitative insights into the system's performance, with ratings offered on a 5-point scale for each parameter.

In the initial evaluations, Our RAG system employed a hybrid retrieval approach combining BM25 (sparse retrieval) and dense retrieval (BAAI/bge-small-en embeddings) using RRF (Top 10) with a chunk size of 400 tokens and an overlap of 200 tokens. Qwen/Qwen2.5-3B-Instruct was used to generate the answers.

Question : What does Anakin Skywalker gift's Padmé Amidala during the night at the Uhmandasee Market and Please Explain its design?

GPT Evaluation:

Final Summary

Aspect	Score	Comments
Faithfulness	2/5	Incorrectly states it was Padmé's braid instead of Anakin's Padawan braid, which changes the core fact.
Relevance	3/5	Stays generally on topic about the gift and its meaning, but factual mistakes weaken it slightly.
Accuracy	2/5	Misidentifies the key object and misses important symbolism details, leading to low accuracy.

Retrieval Time : 50 sec

The results suggest that BM25's high retrieval time, undersized chunks, and an underpowered LLM collectively degraded system performance, emphasizing the need for optimizations in chunking strategy, retrieval methods, and model selection.

In our refined evaluation, the RAG system adopted a dense retrieval-only approach using BAAI/bge-small-en embeddings with larger 1200-token chunks and 600-token overlap.The retrieved passages were processed by the more capable Qwen/Qwen2.5-7B-Instruct model for answer generation. Top 6 Retrieved Chunks were Used.

Question : What is the name of the Neimoidian guard who assists Obi-Wan?

GPT Evaluation:

Quick Summary:

Aspect	Score	Comments
Faithfulness	5/5	No factual errors; matches book
Relevance	5/5	Fully focused on the asked info
Accuracy	5/5	Clear, correct references

Question: What was the cause of the bombing on Cato Neimoidia in Brotherhood?

GPT Evaluation:

Final Summary:

Aspect	Score	Comments
Faithfulness	4/5	Mostly faithful but misses the final confirmed cause.
Relevance	5/5	Very relevant to the bombing topic.
Accuracy	4/5	Correct early plot details but misses the final answer fully.

Question: What is the name of the Jedi Padawan accompanying Obi-Wan in this mission?

GPT Evaluation:

Quick Summary:

Aspect	Score	Comments
Faithfulness	4.5/5	Small technicality about Anakin's Jedi rank
Relevance	5/5	Focused and appropriate
Accuracy	4.5/5	Good, but not formally correct about his Padawan status

Question: What tragedy prompts Obi-Wan to travel to Cato Neimoidia?

GPT Evaluation:

Final Summary:

Aspect	Score	Comments
Faithfulness	5/5	Matches the book details exactly
Relevance	5/5	Focused directly on the asked question
Accuracy	5/5	Correct naming and explanation of the tragedy

Question: What does Anakin Skywalker gifts Padmé Amidala during the night at the Uhmandasee Market and Please Explain its design?

GPT Evaluation:

Final Summary:

Aspect	Score	Comments
Faithfulness	5/5	Matches the story and event perfectly
Relevance	5/5	Sticks exactly to what was asked
Accuracy	5/5	Correct description of design and symbolism

Retrieval Time : 22 sec

The system demonstrated remarkable improvements after optimization, with retrieval time dropping dramatically from 40 seconds to just 22 seconds - a 45% reduction in latency. The larger 1200-token chunks with 600-token overlap provided substantially better context continuity, enabling the Qwen-7B model to generate more accurate and coherent answers.

Denser Retrieval did well as it was able to capture the underlying semantic meaning across different parts of the book instead of relying only on exact word matches.

In "Star Wars: Brotherhood," important details (like character motivations, the cause of the bombing, or the design of Anakin's gift) are often distributed across dialogues, internal thoughts, and actions.

Automated Evaluation using RAGAS

To augment the LLM-based evaluation, we deployed a **GPT-4o-mini model** on Azure and utilized the **RAGAS library** to evaluate our RAG-generated answers. RAGAS provided a systematic and automated evaluation using three key metrics:

1. **Context Recall**: This metric measured how effectively the retrieved documents contributed to the generated response. High context recall indicated that the system utilized the available information comprehensively.

2. **Faithfulness**: This metric assessed the accuracy of the generated response in reflecting the retrieved context without introducing any hallucinated or incorrect details.

3. **Factual Correctness (F1 Mode)**: This metric evaluated the factual alignment of the response with the ground truth, emphasizing precision and recall.

Evaluation Process

- The GPT-4o-mini model was deployed on Azure to serve as the backbone for inference during the evaluation.

- Using RAGAS, we evaluated the system on the aforementioned metrics for a variety of test cases spanning simple to complex queries.
- Scores were computed and analyzed to identify areas of strength and improvement.

```python
from langchain_openai.chat_models import AzureChatOpenAI
from langchain_openai.embeddings import AzureOpenAIEmbeddings
from ragas.llms import LangchainLLMWrapper
from ragas.embeddings import LangchainEmbeddingsWrapper

azure_configs = {
    "base_url": "https://sala-m9pmmei0-eastus2.cognitiveservices.azure.com/",
    "model_deployment": "my-gpt-deployment",
    "model_name": "gpt-4o-mini",
}


azure_llm = AzureChatOpenAI(
    openai_api_version="2023-05-15",
    azure_endpoint=azure_configs["base_url"],
    azure_deployment=azure_configs["model_deployment"],
    model=azure_configs["model_name"],
    openai_api_key="9q7KXIPJUv4MiY5ynSWq0IuBeUvaPWl4MGTnw1qP3MKORkCSmsPfJQQJ99BDACHYHv6XJ3w3AAAAACOGYVRV",
    validate_base_url=False,
)
```

Figure 10: Azure OpenAI Service (Gpt-4o-mini)

```python
from ragas import EvaluationDataset, evaluate
from ragas.llms import LangchainLLMWrapper
from ragas.metrics import LLMContextRecall, Faithfulness, FactualCorrectness

# Initialize query, reference, and RAG model
query = question
reference = "The Neimoidian guard who assists Obi-Wan is Ruug Quarnom."

# Retrieve relevant documents and generate response
relevant_docs = reordered_responses
response = output[0]["generated_text"]

# Create the evaluation dataset for a single query
dataset = [
    {
        "user_input": query,
        "retrieved_contexts": relevant_docs,
        "response": response,
        "reference": reference
    }
]
evaluation_dataset = EvaluationDataset.from_list(dataset)

# Initialize evaluator and evaluate
evaluator_llm = LangchainLLMWrapper(azure_llm)
metrics = [LLMContextRecall(), Faithfulness(), FactualCorrectness()]
result = evaluate(dataset=evaluation_dataset, metrics=metrics, llm=evaluator_llm)

# Print the evaluation results
print(result)
```

Figure 11: Evaluation using RAGAS

Results For Test Queries

```
Query:  What is the name of the Neimoidian guard who assists Obi-Wan?

Response:  The name of the Neimoidian guard who assists Obi-Wan is Ruug. This is evident
from the following excerpt:  "'Don't move, Jedi!' Ruug yelled, and though her
volume increased, her voice carried a level tone. She slowed her approach,
pistol trained on him as she got closer."  Additionally, Ruug is described as a
former special ops Neimoidian guard who helps Obi-Wan during their mission on
Cato Neimoidia.

Golden Answer:  The Neimoidian guard who assists Obi-Wan is Ruug Quarnom.

{'context_recall': 1.0000, 'faithfulness': 1.0000, 'factual_correctness(mode=f1)': 0.4000}
```

Figure 12: Evaluation (Test Query #1)

```
Query:  What was the cause of the bombing on Cato Neimoidia in Brotherhood?

Response:  Based on the information provided in the book "Star Wars: Brotherhood," the
cause of the bombing on Cato Neimoidia remains ambiguous. However, there are
several key points and perspectives that can be inferred:  1. **Conspiracy and
Framing**: Ketar, a Neimoidian official, is presented as someone who might be
involved in a conspiracy. He suggests using evidence to frame the Republic,
indicating that the bombing might be an attempt to incriminate the Republic and
gain political leverage. This perspective implies that the bombing could be a
deliberate act aimed at shifting blame onto the Republic.  2. **Moral Ambiguity
and Political Motives**: The text mentions that Dooku is whispering to the
Neimoidians that the Republic is behind the bombing. This suggests that there
might be political motivations and that the bombing could be a result of
external manipulation or internal political machinations.  3. **Neutral Status
and External Influence**: Cato Neimoidia is described as a neutral world,
despite the ongoing Clone Wars. The fact that the Trade Federation and its
leadership (like Viceroy Nute Gunray) maintained a neutral stance indicates that
the bombing might be an exception to their usual neutrality. This could imply
that the bombing was orchestrated by external forces to disrupt their neutrality
and draw them into the conflict.  4. **Lack of Clear Evidence**: The text also
highlights the uncertainty surrounding the bombing. Obi-Wan is investigating the
incident, and the information provided by Dex, a local informant, suggests that
the bombing might be linked to the Republic, although this is not confirmed. The
lack of clear evidence and the complexity of the situation suggest that the true
cause of the bombing may not be immediately apparent.  In summary, while the
exact cause of the bombing on Cato Neimoidia is not definitively stated, the
narrative presents multiple perspectives and hints at potential external
influences and internal political machinations. The true cause remains unclear,
reflecting the complex and morally ambiguous nature of the events during the
Clone Wars.

Golden Answer:  The cause of the bombing on Cato Neimoidia was a deliberate plot
orchestrated by Asajj Ventress under the orders of Count Dooku and the
Separatists. Their objective was to frame the Republic for the attack
and manipulate the Neimoidians into siding with the Separatists during
the Clone Wars. The bombing was carried out by hiding explosives
within a shipment of Republic aid supplies, making it appear as though
the Republic itself was responsible for the tragedy.  Thus the tragedy
was not an accident or a Republic betrayal, but rather a calculated
act of sabotage and deception masterminded by Dooku, with Ventress
executing his plan.

Result: {'context_recall': 0.0000, 'faithfulness': 0.8947, 'factual_correctness(mode=f1)': 0.3100}
```

Figure 13: Evaluation (Test Query #2)

```
Query:  What is the name of the Jedi Padawan accompanying Obi-Wan in this mission?

Response:  Based on the retrieved information, the Jedi Padawan accompanying Obi-Wan in
this mission is Anakin Skywalker. This can be inferred from several key points:
1. **Page Unknown**: The conversation between Obi-Wan and Anakin discusses plans
and missions, indicating they are working together. 2. **Page Unknown**: Anakin
is mentioned as being present during a situation where guards are attacked,
suggesting he is part of the mission. 3. **Page Unknown**: Anakin is involved in
discussions about podracing and Senator Amidala, further confirming his presence
and role in the mission.  Therefore, the name of the Jedi Padawan accompanying
Obi-Wan in this mission is Anakin Skywalker.

Golden Answer:  The Jedi Padawan accompanying Obi-Wan on this mission is Anakin
Skywalker.

Result: {'context_recall': 1.0000, 'faithfulness': 1.0000, 'factual_correctness(mode=f1)': 0.4000}
```

Figure 14: Evaluation (Test Query #3)

Best Model Selection

Embedding: BAAI/bge-small-en

Chunk Size: 1200

Chunk Overlap: 600

VectorDB: FAISS

Retrievers: Dense Retriever (with k=6 documents)

LLM: Qwen/Qwen2.5-7B-Instruct