A Multimodal Agentic RAG Chatbot Architecture using LangGraph, Qdrant, and Advanced Retrieval Techniques

Hashim Muhammad nadeem[1 [+923112985417]]

FAST National University of Computing and Emerging Sciences 1
i211675@nu.edu.pk

Abstract. Retrieval-Augmented Generation (RAG) has become a cornerstone for building knowledgeable AI systems, grounding large language model (LLM) responses in external data. This paper presents the architecture of a sophisticated multimodal chatbot built upon an Agentic RAG paradigm. Leveraging the LangGraph framework for stateful agent orchestration, the system integrates multimodal capabilities via Google's Gemini models and utilizes Qdrant as its vector store. We detail two novel data ingestion pipelines: an "Agentic Ingestion" process employing vision models for document annotation, and a "Mistral Data Pipeline" using Mistral OCR and Pixtral models for text and image extraction/annotation. The runtime architecture features a router agent directing queries to specialized agents (e.g., Financial, Annual Report, FYP). The Financial Agent demonstrates advanced RAG techniques, including Query Transformation and RAG Fusion, to enhance retrieval relevance and answer quality. Hybrid search combining OpenAI embeddings and BM25, along with semantic chunking and reranking, further refines the retrieval process. This agentic, multimodal, and advanced RAG approach aims to provide highly accurate and contextually relevant responses for complex user queries involving both text and images.

Keywords: Retrieval-Augmented Generation · Agentic AI · Multimodal AI · LangGraph · Qdrant · Query Transformation · RAG Fusion · Data Ingestion · Gemini · OpenAI.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in natural language understanding and generation. However, their knowledge is typically limited to the data they were trained on, leading to potential inaccuracies or "hallucinations" when queried about recent or domain-specific information [1]. Retrieval-Augmented Generation (RAG) addresses this by integrating external knowledge bases into the LLM's generation process [1]. A typical RAG system retrieves relevant document chunks from a vector store based on the user query's semantic similarity and provides these chunks as context to the LLM for generating an informed response.

While effective, basic RAG systems face challenges, particularly with complex queries, multimodal inputs (text and images), and context preservation during data processing [2]. Agentic AI systems, where autonomous agents collaborate to solve problems, offer a promising direction for building more sophisticated RAG applications [3]. Furthermore, advancements in multimodal models like Google Gemini allow for the processing of combined text and image inputs.

This paper describes the architecture of a multimodal Agentic RAG chatbot designed to handle complex queries across different domains (e.g., Financial Reports, Annual Reports, FYP documents). The system is built using LangGraph [4], a library for creating stateful, multi-actor applications, enabling complex agent interactions and cycles. It utilizes Qdrant [5] as a high-performance vector database and leverages both Google Gemini [6] and OpenAI [7] models for multimodal processing and embeddings, respectively.

A key focus of this work is the implementation of advanced RAG techniques during both data ingestion and query processing. We introduce two specialized data ingestion pipelines designed to enhance context preservation and handle multimodal documents. At runtime, the system employs a router agent and specialized agents, with the Financial Agent incorporating Query Transformation [8] and RAG Fusion [9] concepts to improve retrieval accuracy for multifaceted financial queries.

The following sections detail the system's architecture, focusing on data ingestion, agent orchestration, and the core technologies and concepts employed.

2 System Architecture

The overall system architecture, depicted in Fig. 1, consists of two main phases: Data Ingestion and Preprocessing, and Runtime Agent Orchestration and Query Processing.

2.1 Data Ingestion and Preprocessing

Effective RAG relies heavily on how the knowledge base is processed and stored. Traditional methods often split documents into chunks, which can lead to loss of context, hindering retrieval performance [2]. Our system implements two distinct ingestion pipelines located in the `data_ingestion` folder, designed to mitigate this issue and handle multimodal content effectively:

Agentic Ingestion Pipeline: This pipeline draws inspiration from contextual retrieval methods [2]. It employs a vision-capable AI agent (potentially using models like Gemini) to analyze documents during ingestion. The agent's task is to understand the broader context of the document and generate concise, chunk-specific contextual annotations. This annotation, prepended to the chunk content itself, helps the retrieval system understand the chunk's relevance even if the chunk text alone lacks sufficient context. For example, a chunk stating "revenue increased by 5%" might be annotated with "Context: Q3 2023 Financial Report

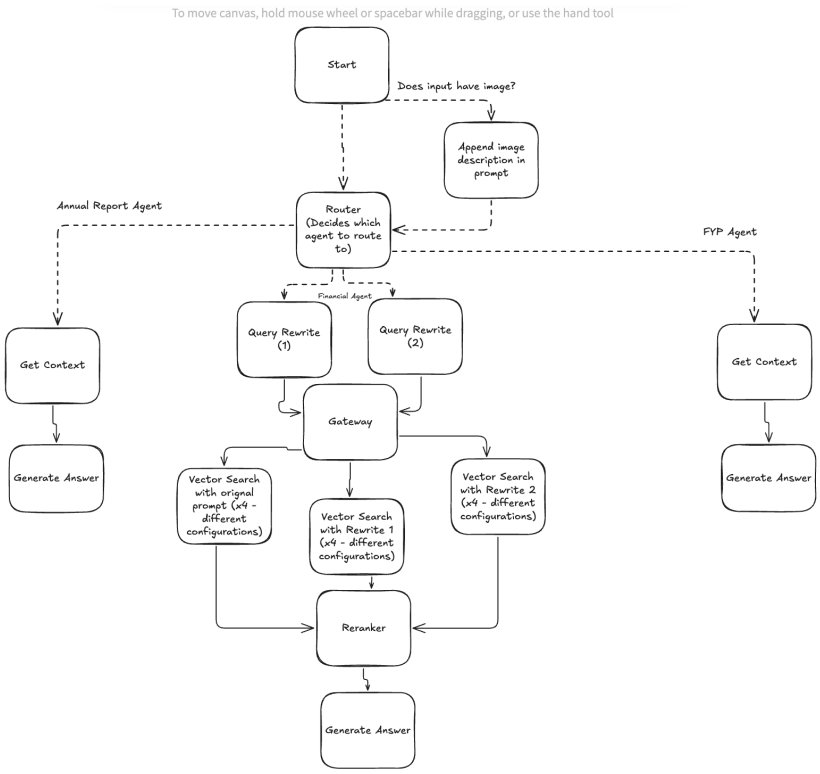

Fig. 1. High-level architecture of the Multimodal Agentic RAG Chatbot.

for ACME Corp, page 12". This approach aims to significantly reduce retrieval failures caused by context loss during chunking.

Mistral Data Pipeline: This pipeline is specifically designed for documents containing a mix of text and images, such as annual reports or technical manuals.

1. **OCR Processing:** It utilizes Mistral OCR to extract text accurately from document pages.
2. **Image Capture:** The pipeline identifies and captures embedded images within the documents.
3. **Image Annotation:** Captured images are sent to a specialized vision model, Pixtral 12b, which generates descriptive text annotations for each image.
4. **Integration:** The text extracted by OCR and the image annotations generated by Pixtral are integrated into the document representation, ensuring that both textual and visual information is captured.

This ensures that queries relating to visual elements (charts, diagrams, photos) can be effectively addressed.

Chunking, Embedding, and Storage: Regardless of the ingestion pipeline used, the subsequent steps are consistent:

1. **Semantic Chunking:** The processed document content (including text, annotations, and image descriptions) is divided into meaningful chunks using a semantic chunker. This approach aims to keep related information together within a single chunk, unlike fixed-size chunking.
2. **Hybrid Embeddings:** To leverage both semantic similarity and keyword matching, we generate hybrid embeddings. This involves:
   - Generating dense vector embeddings using OpenAI's text embedding models (e.g., `text-embedding-ada-002` or newer).
   - Calculating sparse vectors using term frequency-inverse document frequency (TF-IDF) based methods like BM25 [10]. BM25 is particularly effective for queries involving specific keywords, codes, or names [2].
3. **Vector Storage:** Both dense and sparse embeddings, along with the chunk text and associated metadata (e.g., source document, page number, contextual annotations), are stored in a Qdrant [5] vector collection. Qdrant is chosen for its performance, scalability, and support for hybrid search capabilities.

2.2 Agent Orchestration and Query Processing

The runtime query processing is orchestrated using LangGraph [4], allowing for complex, stateful interactions between different AI agents. The flow is initiated when a user submits a query (see Fig. 1):

1. **Start & Multimodal Input Check:** The process begins upon receiving a user query. The system first checks if the user has uploaded an accompanying image.
2. **Image Description (if applicable):** If an image is present, a multimodal model (e.g., Gemini) is used to generate a concise description of the image content. This description is then appended to the original textual query, creating a unified multimodal prompt.
3. **Router Agent:** The potentially augmented query is passed to a Router Agent. This agent, implemented as a LangGraph node, analyzes the query content and decides which specialized agent is best suited to handle it. Based on its analysis, it routes the query to one of the downstream agents: Financial Agent, Annual Report Agent, or FYP Agent.
4. **Agent Execution:** The selected agent takes over processing.

Standard Agent Workflow (Annual Report, FYP Agents): These agents handle queries related to their specific document types (Annual Reports, FYP documents). Their workflow represents a more standard RAG process:

1. **Get Context:** The agent uses the incoming query to perform a hybrid search (semantic + BM25) against the relevant collection(s) in Qdrant. It retrieves the top-K most relevant chunks based on the combined ranking.
2. **Generate Answer:** The retrieved chunks are formatted and passed as context, along with the original query, to an LLM (e.g., Gemini or OpenAI GPT). The LLM synthesizes the information from the context to generate a final answer to the user's query.

Financial Agent Workflow: The Financial Agent is designed for more complex queries often encountered in financial analysis, which might require considering information from multiple perspectives or decomposing the query. It incorporates advanced RAG techniques:

1. **Query Transformation:** Recognizing that the user's initial query might not be optimal for retrieval [8], this agent first employs Query Transformation techniques. As shown in Fig. 1, it generates two rewritten versions of the query (`Query Rewrite (1)`, `Query Rewrite (2)`). These rewrites might involve techniques like:
   - **Hypothetical Document Embeddings (HyDE):** Generating a hypothetical answer and using its embedding for retrieval.
   - **Step-Back Prompting:** Generating a broader, more general question to retrieve foundational context [11].
   - **Sub-Question Generation:** Breaking down a complex query into multiple simpler sub-questions, similar to Multi-Query Retrieval [8].
2. **Gateway:** The original query and the two rewritten queries are passed through a Gateway node. This node likely orchestrates the subsequent parallel searches.

3. **Parallel Vector Search (RAG Fusion Concept):** The system performs multiple vector searches in parallel, implementing a concept similar to RAG Fusion [9]:
   - Search using the **original prompt**.
   - Search using **Rewrite 1**.
   - Search using **Rewrite 2**.

   Each search potentially uses multiple configurations (indicated by "x4 - different configurations" in Fig. 1), which could involve varying the number of results (k), weighting between semantic and keyword search, or applying different metadata filters.
4. **Reranker:** The results (document chunks) from all parallel searches are aggregated. A reranking model (e.g., Cohere Rerank [12] or similar) is then used to re-evaluate the relevance of all retrieved chunks specifically against the *original* user query. Reranking helps to distill the most pertinent information from the larger set of potentially relevant chunks retrieved across different query formulations [2]. It selects the top-N chunks based on the reranker's scores.
5. **Generate Answer:** The final set of top-ranked, reranked chunks is passed as context to the LLM, along with the original query, to generate the definitive answer.

This multi-step process with query transformation, parallel search, and reranking aims to significantly improve retrieval recall and precision for nuanced financial queries compared to a single-pass RAG approach.

3 Core Technologies and Concepts

Several key technologies and advanced AI concepts underpin the described architecture:

3.1 LangGraph Framework

LangGraph [4] provides the foundation for orchestrating the agents. It extends the popular LangChain library by allowing the definition of agent workflows as graphs. This is crucial for implementing cycles (e.g., for self-correction or iterative refinement), conditional routing (as seen with the Router Agent), and managing the state across complex interactions involving multiple LLM calls and tool uses.

3.2 Multimodal Capabilities (Gemini)

Google's Gemini models [6] are employed for their native multimodal understanding capabilities. This allows the system to:

- Process user inputs containing both text and images.
- Generate textual descriptions of images during input processing.

- Potentially be used within the Agentic Ingestion pipeline for analyzing document layouts and content.
- Generate final responses that can potentially reference both textual and visual information retrieved from the context.

3.3 Vector Storage (Qdrant)

Qdrant [5] serves as the vector database. Its key features relevant to this project include:

- High performance for similarity search on large datasets.
- Support for storing both dense (OpenAI) and sparse (BM25) vectors, enabling efficient hybrid search.
- Rich filtering capabilities based on metadata stored alongside vectors (e.g., document source, type, annotations).
- Scalability to handle growing knowledge bases.

3.4 Advanced RAG Techniques

Beyond standard RAG, the system incorporates several advanced techniques:

Agentic / Contextual Ingestion: As described in Section 2.1.1, this method leverages an AI agent during ingestion to add contextual summaries to chunks, directly addressing the context loss problem in traditional RAG [2]. This preprocessing step enhances the retrievability of isolated chunks.

Query Transformation: Employed by the Financial Agent, this involves using an LLM to rewrite the user's query into one or more new queries that are potentially better suited for retrieval [8]. Techniques like step-back prompting or generating multiple perspectives help uncover relevant information that the original query phrasing might miss.

RAG Fusion: The parallel search strategy using the original and transformed queries, followed by reranking, embodies the core idea of RAG Fusion [9]. By retrieving documents based on multiple query variations and intelligently combining the results (using rank fusion algorithms like Reciprocal Rank Fusion, or simply reranking the combined set as done here), the system casts a wider net and improves the chances of finding the most relevant information.

Hybrid Search: Combining dense vector search (capturing semantic meaning via OpenAI embeddings) with sparse vector search (capturing keyword relevance via BM25) provides a more robust retrieval mechanism than either method alone [2]. Qdrant facilitates this combination.

Reranking: After an initial retrieval stage that might return a large set of candidate chunks (especially with RAG Fusion), a dedicated reranking model provides a more fine-grained relevance assessment relative to the original query [2,**?**]. This filters noise and ensures the final context passed to the LLM is highly relevant, improving generation quality and efficiency.

4 Implementation Details

The system is implemented using Python, leveraging the following key libraries and models:

- **Orchestration:** LangChain and LangGraph.
- **LLMs/Multimodal Models:** Google Gemini family (e.g., Gemini Pro, potentially Gemini Vision Pro), OpenAI GPT family (e.g., GPT-4, GPT-3.5-turbo).
- **Embeddings:** OpenAI text embedding models (e.g., `text-embedding-3-small/large`).
- **Vector Store:** Qdrant, accessed via its Python client.
- **Ingestion Specifics:** Mistral OCR, Pixtral 12b (for Mistral pipeline), potentially custom agents using Gemini/GPT for the Agentic Ingestion pipeline. Semantic chunking libraries (e.g., based on sentence transformers or specific LangChain chunkers). BM25 implementation (e.g., from `rank_bm25` library).
- **Reranker:** Integration with a reranking model API (e.g., Cohere Rerank).

The "x4 different configurations" mentioned for vector search in the Financial Agent likely refers to programmatic variations in search parameters, such as the number of results requested (`k`), the α value controlling the balance in hybrid search, or specific metadata filters applied during the search query to Qdrant.

5 Conclusion and Future Work

This paper presented the architecture of a sophisticated multimodal Agentic RAG chatbot built on LangGraph. By combining specialized data ingestion pipelines that enhance context preservation (Agentic/Contextual Ingestion, Mistral Pipeline) with advanced runtime techniques like Query Transformation, RAG Fusion, hybrid search, and reranking within a multi-agent framework, the system aims to deliver superior performance on complex, multimodal queries compared to traditional RAG approaches. The use of LangGraph enables the complex orchestration required for these interacting components.

Future work could involve:

- **Rigorous Evaluation:** Conducting quantitative evaluations using standard RAG metrics (e.g., context precision/recall, faithfulness, answer relevance) on domain-specific benchmarks.
- **Adaptive Routing:** Enhancing the Router Agent to learn or adapt its routing strategy based on user feedback or conversation history.

- **Agent Memory:** Incorporating more sophisticated memory mechanisms within agents to handle longer conversations more effectively.
- **Tool Use Expansion:** Equipping agents with more tools, such as calculators, data analysis libraries, or external APIs, for more complex reasoning tasks.
- **Optimization:** Investigating cost and latency trade-offs, potentially using techniques like prompt caching [2] where applicable.

This agentic architecture provides a flexible and powerful foundation for building next-generation RAG systems capable of handling diverse and complex information needs.

References

1. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.t., Rocktäschel, T., Riedel, S., Kiela, D.: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In: Advances in Neural Information Processing Systems (NeurIPS) 33 (2020)
2. Anthropic: Introducing Contextual Retrieval. Anthropic Blog (2024). `https://www.anthropic.com/news/introducing-contextual-retrieval`
3. Xi, Z., Chen, W., Wang, X., Dou, Y., Yang, C., Wang, Z., Qiu, X.: The Rise and Potential of Large Language Model Based Agents: A Survey. arXiv preprint arXiv:2309.07864 (2023)
4. LangChain: LangGraph Documentation. `https://python.langchain.com/docs/langgraph/`. Accessed [Insert Access Date Here]
5. Qdrant: Qdrant Vector Database. `https://qdrant.tech/`. Accessed [Insert Access Date Here]
6. Gemini Team, Google: Gemini: A Family of Highly Capable Multimodal Models. arXiv preprint arXiv:2312.11805 (2023)
7. OpenAI: Text embedding models. OpenAI Documentation. `https://platform.openai.com/docs/guides/embeddings`. Accessed [Insert Access Date Here]
8. LangChain Blog: Query Transformations. (2023). `https://blog.langchain.dev/query-transformations/`
9. LangChain Blog: RAG Fusion. (Referencing the concept, often linked to the Multi-Query approach with reranking). Accessed [Insert Access Date Here]. (Note: A direct canonical "RAG Fusion" blog post might need specific finding, this refers to the concept discussed in their Multi-Query/Reranking implementations).
10. Robertson, S., Zaragoza, H.: The Probabilistic Relevance Framework: BM25 and Beyond. Found. Trends Inf. Retr. 3(4), 333–389 (2009)
11. Zheng, H., Zaheer, M., Wang, Z., Chen, M., Singh, A., Hsieh, C.J.: Generating Step-by-Step Thinking for Large Language Models. arXiv preprint arXiv:2305.04155 (2023)
12. Cohere: Rerank Documentation. `https://docs.cohere.com/reference/rerank`. Accessed [Insert Access Date Here]