Detailed Design Document: Memory-Persistent AI Research Assistant

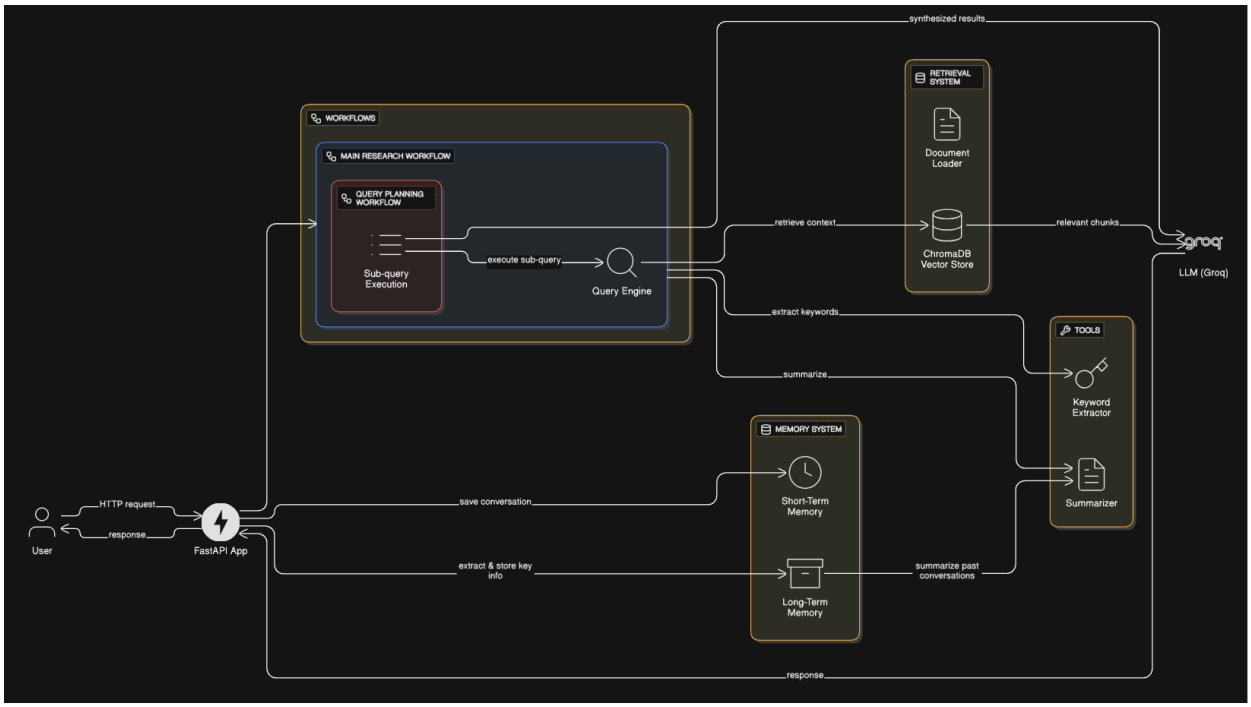
1. Introduction

This document outlines the architecture and design of the Memory-Persistent AI Research Assistant. The primary goal of this project is to create an intelligent assistant that can answer questions about a given document while maintaining conversational context across multiple sessions. The system is designed to be modular, scalable, and maintainable, leveraging modern AI and web technologies.

2. System Architecture

The system is designed around a modular architecture, with a clear separation of concerns between the different components. The core of the application is built on LlamaIndex, which provides a powerful framework for building LLM-powered applications. The entire system is exposed as a FastAPI application, making it easy to interact with via a REST API.

High-Level Diagram



3. Component Deep Dive

3.1. FastAPI Application (`src/app.py`)

The FastAPI application serves as the main entry point to the system. It exposes a `/query` endpoint that accepts user queries and returns the assistant's response. It is responsible for initializing the necessary components, such as the workflows, memory system, and query engines.

3.2. Workflows (`src/workflows/`)

The core logic of the application is encapsulated in LlamaIndex workflows. This provides a structured and modular way to define the steps involved in processing a query.

- `MainResearchWorkflow`: This is the primary workflow that orchestrates the entire

process. It receives a query, assesses its complexity, and then routes it to the appropriate processing path.
- `QueryPlanningWorkflow`: This workflow is responsible for handling complex queries. It breaks down a complex question into smaller, more manageable sub-queries, executes them, and then synthesizes the results into a single, coherent answer.

3.3. Memory System (`src/memory/`)

The memory system is a crucial component that allows the assistant to maintain context across conversations.

- `ShortTermMemory`: This component stores the history of the current conversation, allowing the assistant to understand follow-up questions and maintain a natural conversational flow.
- `LongTermMemory`: This component extracts and stores key facts, concepts, and research topics from conversations. This allows the assistant to build a persistent knowledge base over time.

3.4. Retrieval System (`src/retrieval/`)

The retrieval system is responsible for finding relevant information in the source document.

- `DocumentLoader`: This component is responsible for loading the `adobe-annual-report.pdf` and preparing it for indexing.
- `Vector Store (ChromaDB)`: The document is converted into vector embeddings and stored in ChromaDB. This allows for efficient similarity-based searches.
- `QueryEngine`: This component takes a query, converts it into an embedding, and then searches the vector store for the most relevant document chunks.

3.5. Tools (`src/tools/`)

The system includes a set of tools that provide additional capabilities:

- `KeywordExtractor`: This tool uses YAKE and KeyBERT to extract important keywords from a given text.
- `Summarizer`: This tool can be used to generate concise summaries of documents or conversations.

4. Design Decisions and Rationale

Component/Decision	Rationale
FastAPI	FastAPI was chosen for its high performance, asynchronous support (which is crucial for I/O-bound operations like LLM calls), and automatic generation of interactive API documentation.
LlamaIndex Workflows	The workflow engine in LlamaIndex provides a powerful and flexible way to orchestrate complex, multi-step processes. It allows for a clear separation of concerns and makes the code more modular and easier to maintain.
Dual Memory System	A dual memory system was implemented to address the different needs of conversational context and long-term knowledge retention. Short-term memory is essential for natural conversation, while long-term memory allows the assistant to learn and grow over time.
ChromaDB	ChromaDB was selected as the vector store for its simplicity, ease of use, and local-first, persistent storage. This makes it ideal for a self-contained research assistant.
Groq for LLM	The Groq API provides extremely fast inference speeds, which is critical for a real-time conversational agent. This ensures a smooth and responsive user experience.

Docker	Docker and Docker Compose are used to containerize the application, ensuring a consistent and reproducible environment for both development and deployment. This simplifies dependency management and makes it easy to run the application on any machine.

5. Data Flow

1. A user sends a POST request to the `/query` endpoint with a question.
2. The `MainResearchWorkflow` is initiated.
3. The workflow assesses the query's complexity.
    - If the query is simple, it is sent directly to the RAG query engine.
    - If the query is complex, it is passed to the `QueryPlanningWorkflow`.
4. The `QueryPlanningWorkflow` breaks the complex query into sub-queries.
5. Each sub-query is executed by the RAG query engine.
6. The results of the sub-queries are synthesized into a single, comprehensive answer by the LLM.
7. The final response is returned to the user.
8. The conversation (query and response) is saved to the short-term memory.
9. Key information is extracted and stored in the long-term memory.

6. Future Improvements

- Support for Multiple Documents: The system could be extended to support multiple documents, allowing users to ask questions across a larger corpus of information.
- More Sophisticated Memory: The memory system could be enhanced with more advanced techniques, such as summarization of past conversations or the ability to forget irrelevant information.
- Integration with Other Tools: The assistant could be integrated with other tools, such as a web search tool, to provide more comprehensive answers.
- User-Specific Memory: The memory system could be personalized to each user, allowing the assistant to remember user-specific preferences and context.