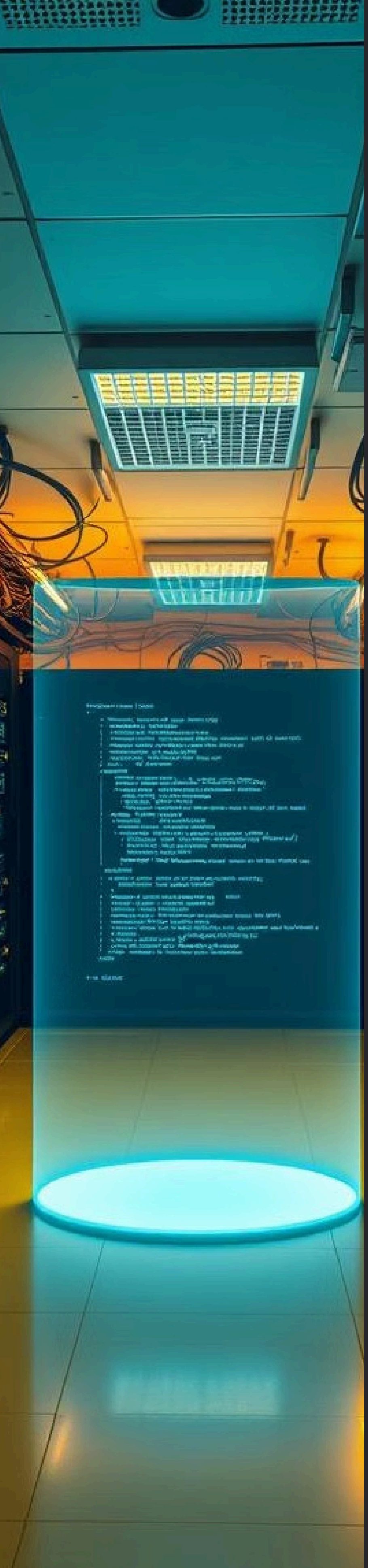


End-to-End Query Flow

Our comprehensive system seamlessly orchestrates the journey from document ingestion to a generated, evaluated, and logged answer, ensuring a robust and transparent conversational AI experience.

Document Ingestion

Documents are uploaded and ingested into the PGVector database via the Docling Loader, forming the knowledge base for retrieval.

User Query Initiation

Users initiate the process by submitting questions through the OpenAI-compatible `/chat/completions` API endpoint.

Query Refinement

The user's query undergoes intelligent refinement and orchestration, powered by CrewAI, to optimize it for subsequent steps.

Advanced Retrieval

The refined query activates the multi-stage retriever (Vector, BM25, Fusion, Reranker) to secure the most relevant context chunks.

Memory Integration

Relevant conversational memory is added, providing crucial context for pronoun resolution and coreference support within the LLM prompt.

Prompt & LLM Processing

A comprehensive prompt, including context and memory, is assembled and passed to the Ollama Large Language Model for grounded answer generation.

Answer & Output

The system returns the generated answer, complete with source citations and preliminary RAGAS evaluation scores for transparency.

Full Lifecycle Logging

Every step of the query flow is meticulously logged via Phoenix, enabling precise trace replay, debugging, and continuous performance monitoring.

1. Custom Open WebUI Setup

Our implementation features a customized Open WebUI, directly integrated with a central Flask API to manage all Retrieval-Augmented Generation (RAG) processes, effectively disabling local RAG functionalities.

Open WebUI

The user-facing interface for interaction, connecting directly to the custom Flask API.

Flask API

Serves as the central backend, handling all incoming requests from the WebUI, including document uploads and user queries.

RAG Pipeline Orchestration

The API orchestrates the entire RAG pipeline, managing ingestion, advanced retrieval (Vector + BM25 + Reranker), and answer generation.

PGVector & Ollama LLM

Leverages PGVector for knowledge base storage and Ollama as the Large Language Model for generating grounded responses.

Response to WebUI

The final generated answer and source citations are seamlessly returned to the Open WebUI for display.

This architecture ensures a robust, scalable, and fully controlled RAG environment tailored to specific needs.

2. Document Ingestion and Retrieval Pipeline

Our document ingestion and retrieval pipeline is designed for efficiency, accuracy, and semantic preservation, ensuring that your data is always ready for intelligent retrieval. This sophisticated process transforms raw documents into structured, searchable information within your system.

API Upload & Ingestion

Documents are securely uploaded through a dedicated API endpoint (`/tools/upload-docs/files`). This initial step handles various file formats and ensures robust data intake.

Docling Conversion

Once uploaded, documents, particularly PDFs, are converted into a standardized markdown format using **Docling**. This conversion preserves formatting and structure, making the content machine-readable and semantically coherent.

LlamaIndex Document Wrapping

The converted markdown content is then wrapped as **LlamaIndex Documents**. This process enriches each document with crucial metadata, such as source, creation date, and custom tags, enabling more granular retrieval.

Variable Chunking

Instead of fixed-size chunks, our pipeline employs **variable chunking**. This intelligent segmentation preserves semantic units within the document, ensuring that contextually related information remains together, which significantly improves retrieval accuracy.

PGVector Storage & Ollama Embeddings

The processed chunks are stored in **PGVector**, a PostgreSQL extension for vector similarity search. Each chunk is transformed into a high-dimensional embedding using **Ollama**, a powerful open-source language model, enabling efficient semantic search.

3. Retriever Logic: Precision Search & Ranking

Advanced retriever logic employs a multi-faceted approach to ensure highly accurate and contextually relevant document retrieval. By combining different retrieval techniques and a sophisticated re-ranking mechanism, we maximize the precision of search results.

Query Initiation

Every search begins with a user query, which is then processed by our intelligent retrieval system to identify the most relevant document chunks.

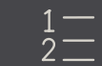
Fusion Retrieval

The retriever executes two parallel search methods:

- **Vector Retrieval:** Identifies semantically similar chunks using high-dimensional embeddings.
- **BM25 Retrieval:** Matches keywords for sparse, exact term relevance.

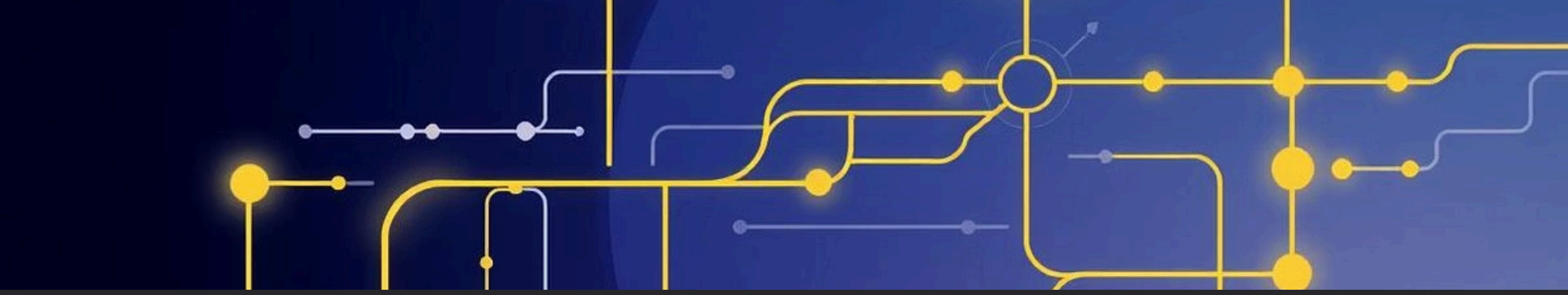
Reciprocal Rank Fusion (RRF)

Results from both retrieval methods are dynamically merged using **Reciprocal Rank Fusion (RRF)**, which assigns higher scores to documents that rank well in both semantic and keyword searches.



Cross-Encoder Reranking

Finally, a powerful **Cross-Encoder Reranker** re-scores the merged results for enhanced precision, ensuring the most contextually accurate and relevant information is presented first.



4. Pipeline: Retrieval + Prompt Assembly + LLM

This stage orchestrates the seamless flow from a user's query to a generated answer, integrating retrieved information and conversational history to construct precise prompts for the Language Model.

Query & Context Retrieval

The process begins with a user query, which triggers the retrieval of the most relevant 'top-k' context chunks from our vectorized knowledge base.

Dynamic Context Building

Retrieved chunks are intelligently assembled into a character-limited 'CONTEXT' block, optimized for efficient processing by the Language Model.

Conversation History

Relevant 'CHAT HISTORY' is integrated from memory, providing the LLM with crucial conversational context from prior Q&A interactions.

Strict RAG Prompting

The final prompt is rigorously assembled, instructing the RAG assistant to strictly use the provided CONTEXT and HISTORY, and to state if information is insufficient.

Ollama LLM Processing

The complete prompt is then passed to the Ollama Large Language Model, leveraging its powerful capabilities for generating accurate and relevant responses.

Answer & Reference Output

Finally, the system collects the generated answer along with all relevant source references, ensuring transparency and verifiability of the information provided.

5. Chat Completion API

The Chat Completion API provides an OpenAI-compatible endpoint (`/v1/chat/completions`) for seamless integration with existing Open Web UI, orchestrating the final stages of interaction.

Extract Query & Session ID

The API initiates by precisely extracting the user's query and associated session ID from the incoming request.

Log User Query

Each user query is securely logged into an in-memory buffer, preserving conversational context for subsequent interactions.

CrewAI Refinement

The query and retrieved context undergo advanced refinement and decision-making processes powered by CrewAI for optimal prompt construction.

Store Assistant Response

The generated assistant's response is stored back into memory, ensuring continuity and recall within the user's session.

Return Structured JSON

The final output is delivered as structured JSON, including the answer, source citations, and comprehensive RAGAS metrics for quality assurance.

Streaming Support

The API supports both traditional non-streaming responses and Server-Sent Events (SSE) for real-time, token-by-token output.

6. Evaluation & 7. Observability

Beyond just functionality, our system prioritizes continuous improvement through robust evaluation and comprehensive observability.

6. Evaluation: Ensuring Quality Responses

The Ragas Evaluator computes critical metrics to ensure the accuracy, relevance, and factual grounding of every generated response.

Faithfulness

Confirms responses are factually accurate and directly supported by the retrieved source documents.

Relevance

Measures how well the generated answers address the user's query and stay on topic.

Correctness

Verifies the overall factual correctness and absence of contradictions in the response.

These metrics are vital for maintaining high standards and user trust in the system's output.

7. Observability: Full Pipeline Visibility

Integration with the Phoenix Dashboard via OpenTelemetry provides end-to-end logging and real-time monitoring of every step within the pipeline.

Query Refinement

Logs how initial user queries are processed and optimized.

Vector Retrieval

Captures the performance and results of semantic searches.

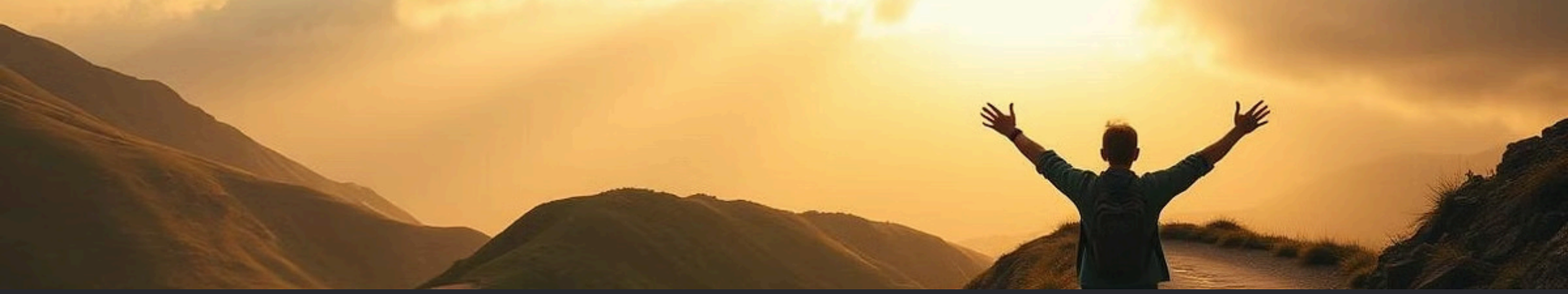
BM25 Retrieval

Records the outcomes of keyword-based search operations.

LLM Completion

Tracks the final prompt assembly and language model response generation.

This allows for precise trace replay, efficient debugging, and continuous performance monitoring, ensuring optimal system health.



Challenges Faced During RAG Project

Navigating the complexities of a RAG implementation brought forth several hurdles, each overcome with strategic adjustments and innovative solutions.

Ollama Embeddings Performance

Challenge: Initial slowness with Ollama embeddings impacted overall system responsiveness.
Solution: Implemented parallel processing to significantly boost embedding generation speed.

CPU-Only Setup Limitations

Challenge: Operating in a CPU-only environment led to performance bottlenecks, particularly under heavy load.
Solution: Used smaller embedding and LLMs

Open WebUI Integration

Challenge: Faced integration issues with Open WebUI, including document upload overrides and local RAG conflicts, along with streaming problems.
Solution: Customized configurations to ensure seamless functionality and address streaming inconsistencies.

Retrieval Quality with Fixed Chunking

Challenge: Poor retrieval quality resulted from using a fixed chunking strategy, which missed contextual nuances.
Solution: Transitioned to a more sophisticated variable chunking approach for improved relevance and accuracy.

Future Optimizations

Our roadmap includes key enhancements to further refine the RAG system's capabilities, focusing on robustness, versatility, and user experience.

Guardrails for Safety

Implement robust content filters to detect and prevent hate speech, personally identifiable information (PII), and other vulgar content.

Multi-modal RAG

Expand retrieval capabilities to include not just text, but also images, tables, and audio/video, creating a richer knowledge base.

Smarter Chunking

Develop advanced chunking strategies utilizing semantic splitting to ensure more coherent and contextually relevant document segments.

Advanced Rerankers

Integrate sophisticated reranking algorithms to further refine search results, significantly improving the relevance of retrieved information.

Distributed GPU Setup

Transition to a distributed GPU infrastructure to enhance processing power and ensure seamless scalability for growing demands.

User Feedback Loop

Establish a continuous user feedback mechanism to gather insights and iteratively tune the system for optimal performance and user satisfaction.