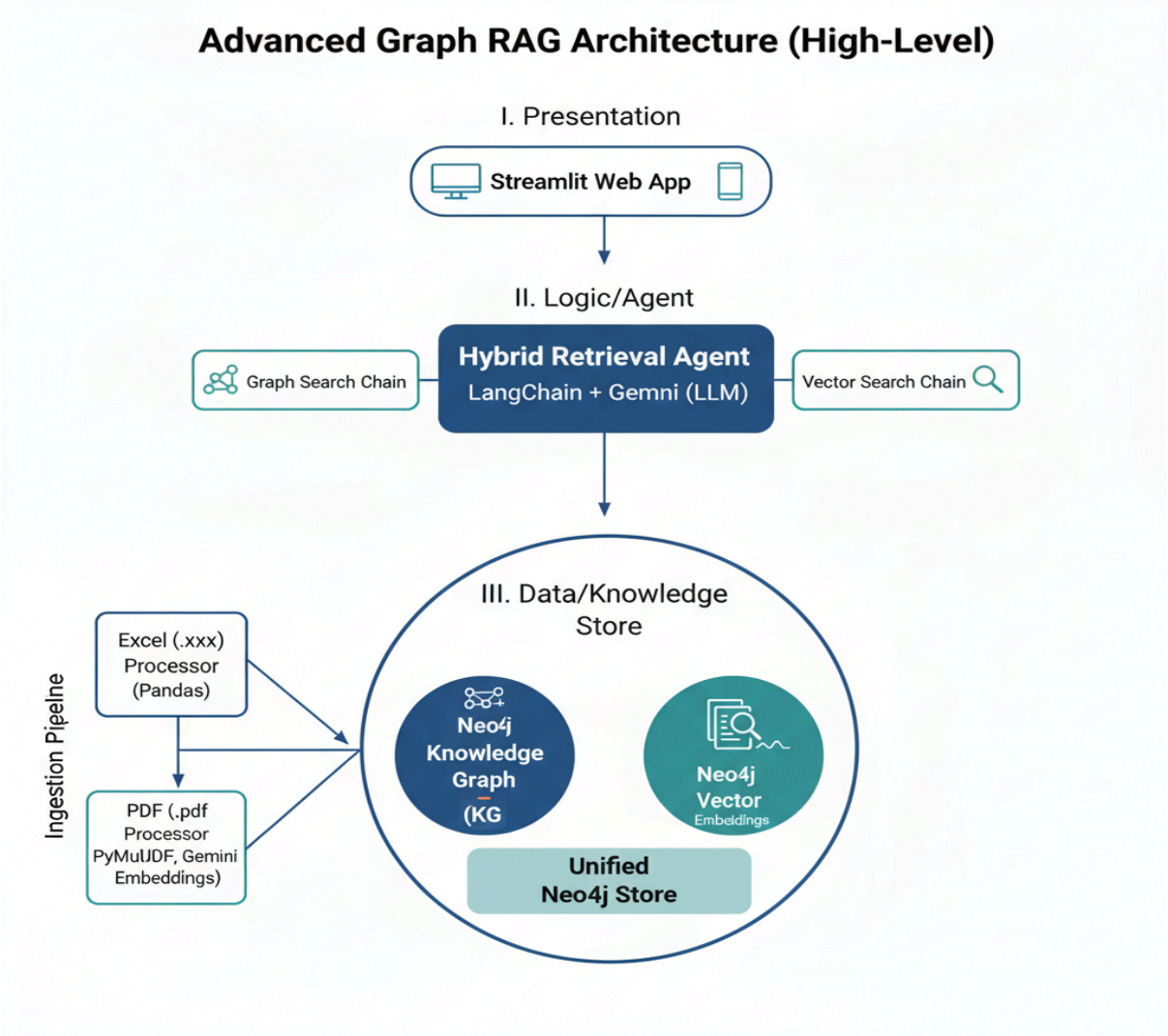
Documentation And Architecture Diagram of Advanced Graph RAG System (Gemini + Neo4j Hybrid Agent)

Problem statement :
Use Knowledge Graph databases like NeoJ4 and create proper graphical relationships among the data given (pdf + excel files) and create an advanced RAG out of it

Solution:
This document outlines the architecture, setup, and functionality of an **Advanced Graph RAG (Retrieval-Augmented Generation) System**. This solution leverages the power of **Neo4j** (as a Knowledge Graph and Vector Database) and the **Gemini API** via LangChain to create an intelligent AI agent capable of answering complex queries that require both structured/relational data and unstructured context.

1. Project Overview and Architecture

The system is designed as a **Hybrid Retrieval Pipeline** that intelligently routes a user's question to the most appropriate data store: the Knowledge Graph for facts and relationships, or the Vector Store for contextual summaries.



Core Architecture Components

Phase	Component	Technology/Tool	Purpose
Ingestion	**Knowledge Graph (KG) & Vector Index Creation**	ingest_utility.py, pandas, pdfplumber, langchain-neo4j	Processes Excel (.xlsx) into **Structured Nodes/Relationships** and PDFs (.pdf) into **Text Chunks with Vector Embeddings** in Neo4j.
Retrieval	**Intelligent Query Router**	**Gemini (LLM)**, langchain-core	Classifies user questions as **"GRAPH"** (relational/metric-based) or **"VECTOR"** (contextual/summary-based).
	Graph Search	GraphCypherQAChain, **Neo4j**	Translates "GRAPH" questions into Cypher queries for precise retrieval of facts and relationships.
	Vector Search	Neo4jVector, **Gemini Embeddings**	Performs semantic search on "VECTOR" questions, retrieving relevant text chunks from the PDF corpus.
Generation	**Final Answer Synthesis**	**Gemini (LLM)**, langchain-core	Generates the final, grounded natural language answer using the retrieved context from either the Graph or Vector search.

---

2. Setup and Prerequisites

2.1 Dependencies

The following packages are required to run the application.

```
pandas
openpyxl
pdfplumber
streamlit
langchain
langchain-community
langchain-neo4j
langchain-google-genai
neo4j
python-dotenv
requests
langchain_experimental
```

2.2 Environment Variables (`.env`)

You must configure your credentials and model names in a `.env` file at the root of the project.

Variable	Description	Source
`GEMINI_API_KEY`	Your API key for Google's Gemini models.	Google AI Studio
`GOOGLE_API_KEY`	(Alias for `GEMINI_API_KEY`)	Google AI Studio
`NEO4J_URI`	The connection URL for your Neo4j database (e.g., AuraDB instance).	Neo4j AuraDB Console
`NEO4J_USERNAME`	Usually `neo4j`	Neo4j AuraDB Console

NEO4J_PASSWORD	The password for your Neo4j instance.	Neo4j AuraDB Console
LLM_MODEL	The model used for reasoning, routing, and generation (e.g., gemini-2.5-flash).	Gemini API
EMBEDDING_MODEL	The model used for generating vector embeddings (e.g., text-embedding-004).	Gemini API

---

3. Application Workflow (app.py)

The main application is a Streamlit interface that orchestrates the Hybrid RAG pipeline.

3.1 Initialization and Caching

The initialize_rag_components() function uses @st.cache_resource to ensure the expensive operations—like connecting to the LLM/Neo4j and refreshing the graph schema—only happen once when the application starts.

3.2 Graph Search Chain

The system sets up the GraphCypherQAChain using **Gemini** to translate natural language into the Neo4j query language, **Cypher**.

The custom cypher_generation_prompt provides specific examples to the LLM, guiding it to generate precise Cypher based on the existing graph schema (e.g., Company, Metric, HAS_METRIC).

3.3 Vector Search Retriever

The setup_vector_retriever() function connects to the **pdf_vector_index** in Neo4j, using the Neo4jVector class with the specified Gemini embeddings_instance.

3.4 Intelligent Query Router

This is the core of the hybrid solution. The intelligent_query_router(question) function:

1. Feeds the user's question into a specific **Gemini prompt**.
2. The prompt forces Gemini to respond with only one of two keywords: **"GRAPH"** or **"VECTOR"**.
3. This single-word classification determines which retrieval path is taken in the next step.

3.5 Hybrid RAG Execution (run_hybrid_query)

This function executes the chosen retrieval path:

- **If 'GRAPH':** It runs the graph_qa_chain. This returns a direct, factual answer and exposes the intermediate **Cypher query** for transparency. It includes a fallback to Vector Search if the Graph Query fails.
- **If 'VECTOR':** It uses the vector_retriever to fetch the top 3 relevant Chunk nodes from the PDF vector index. It then passes these chunks as **context** to a final RAG prompt for Gemini to generate a summarized, contextual answer.

---

4. Data Ingestion Utility (ingest_utility.py)

This script handles the ingestion pipeline, intelligently processing both Excel and PDF files into the Neo4j database.

4.1 Data Cleaning and Setup

- clean_value(value): Normalizes data types, converting strings to floats or dates where appropriate, and handling empty/NaN values.
- clear_data_by_source(file_name): Ensures idempotency by deleting any pre-existing data tied to the uploaded file name before ingestion.

4.2 Excel Ingestion (ingest_excel_intelligent)

This function adapts its ingestion logic based on the Excel structure:

- **Simple (Key-Value):** If columns are , it treats the file as a list of financial metrics (Metric nodes) and links them to a Company node via the :HAS_METRIC relationship.
- **Complex (Tabular):** If the data is highly tabular (e.g., Inventory, Books, Invoices), it creates detailed entity nodes (Book, InventoryItem, etc.) for each row.
    - It also uses helper functions (create_book_relationships) to extract and create **secondary entities** like Author, Publisher, and Store, connecting them relationally.
    - It calculates and stores **aggregate metrics** (e.g., "Total Opening Stock," "Average Price") as a type of Metric node, allowing the Graph Search to answer summary-based numerical questions.

4.3 PDF Ingestion (ingest_pdf_intelligent)

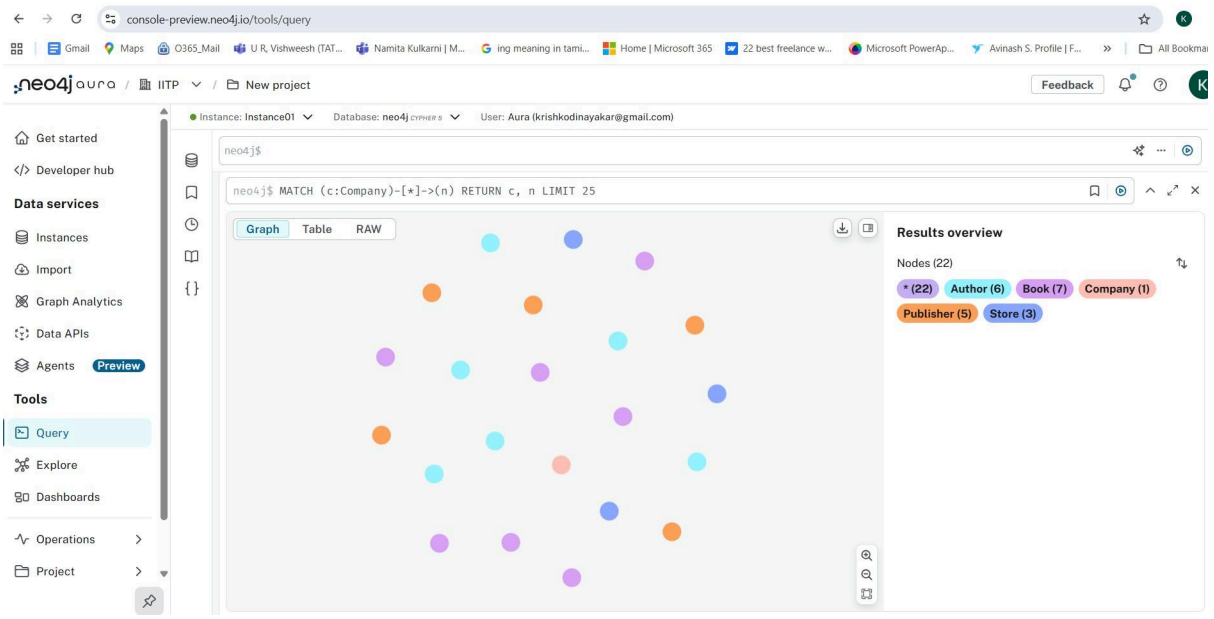
This function performs two critical tasks:

1. **Structured Extraction:** It uses Python's re module (extract_invoice_data) to parse key-value pairs (Invoice No, Total Amount, Date) from the raw text, creating structured Invoice and Customer nodes.
2. **Unstructured Chunking & Vectorization:** It uses PDFPlumberLoader for text extraction, RecursiveCharacterTextSplitter to create small, overlapping text chunks, and the **Gemini Embeddings** model to calculate vector embeddings, which are stored in the **Chunk** nodes in the Neo4j vector index.

5. Usage Guide (Streamlit UI)

1. **Run the application: (refer User Guide)**

Final Output in Neo4j Browser: (explained in User Guide



****