**Legal Query Agent**

**Overview**

The Legal Query Agent is designed to process legal queries, categorize them into specific domains, retrieve relevant legal documents, and generate concise responses based on the retrieved information. The agent employs a multi-step processing pipeline using **Large Language Models (LLMs), Embedding Models, and Information Retrieval Techniques**.

Built using technologies like Llama-Index , LangGraph

**Key Components**

**1. Language Models and Embeddings**

* The agent uses **Ollama Llama3.2 (3B)** for query processing and response generation.
* The embedding model **Jina Embeddings v2** is utilized for document retrieval, leveraging a local Ollama server.

**2. Retrieval Mechanism**

The retrieval system is designed to fetch relevant legal information efficiently:

* **Vector Store Index**: Documents are indexed using embeddings for **semantic search**.
* **BM25 Retriever**: A traditional keyword-based retrieval mechanism improves **precision for legal texts**.
* **Query Fusion Retriever**: Combines **semantic and keyword-based retrieval** for optimal results.

**3. State Management & Processing Workflow**

The agent is structured as a **stateful pipeline** using LangGraph:

**Step 1: Query Routing**

* The route function categorizes queries into **'litigation'** or **'financial'** domains.
* A prompt instructs the LLM to determine the category.
* The output is strictly one of the two categories.

**Step 2: Document Retrieval**

* The retrieve function selects the appropriate retriever based on the routed category.
* The retriever fetches the most relevant legal documents.

**Step 3: Response Summarization**

* The summarize function generates a factual response based on retrieved legal texts.
* The response strictly adheres to the **provided legal context** without speculation.

**Graph-Based Execution Flow**

The agent is implemented using **LangGraph**, ensuring a structured and efficient flow:

1. **Router Node**: Categorizes the query (litigation or financial).
2. **Retriever Node**: Fetches relevant legal documents.
3. **Summarizer Node**: Generates a factual response.
4. **Edges**:
   * START → Router → Retriever → Summarizer → END

**Architecture Diagram:**

A purple rectangular sign with black text

AI-generated content may be incorrect.

**Improvements:**

**1. Fine-Tuning Embeddings**

* **The current model uses Jina Embeddings v2, but fine-tuning the embeddings on legal documents can improve retrieval accuracy.**
* **Fine-tuning would allow the model to better capture domain-specific terminology, such as legal clauses, case law references, and financial regulations.**
* **This could be done using supervised contrastive learning, where positive and negative examples are generated from case laws and legal statutes.**

**2. LLM-Based Parsing of PDFs**

* **Instead of relying on simple text extraction methods, LLMs can be leveraged to parse and structure PDF content more effectively.**
* **Challenges with PDF parsing:**
  + **PDFs often contain tables, footnotes, multi-column text, and scanned images that traditional extractors struggle with.**
  + **Using an LLM with OCR and structured parsing capabilities can help extract sections, headings, and legal references more accurately.**

**3. Use of Reranker**

* **The current retrieval mechanism primarily relies on BM25 and embedding-based similarity search, but adding a reranker can improve relevance.**
* **Rerankers, such as Cohere Rerank or LlamaIndex Rerankers, use LLMs to score retrieved documents based on query relevance.**
* **A reranker can ensure that legally precise and contextually relevant documents are prioritized over semantically similar but less useful documents.**

**4. Using a Better Embedding Model and LLM**

* **Currently, the system relies on Llama3 (3B), which is relatively small for complex legal reasoning.**
* **Switching to larger LLMs like Llama3 (8B) or Mixtral can significantly enhance legal reasoning and summarization capabilities.**
* **For embeddings, models like BGE-M3 or OpenAI's text-embedding-3-large could provide better vector representations of legal documents.**

**5. Adding Metadata While Indexing**

* **Indexing documents with metadata such as document type, date, legal jurisdiction, and section headings can improve retrieval.**
* **Instead of treating all legal texts as raw chunks, adding structured metadata allows for better filtering and retrieval based on legal context.**
* **Example: If a query asks about "Supreme Court rulings on taxation," metadata-based retrieval can ensure only Supreme Court cases related to taxation are fetched.**

**6. Using Advanced RAG Techniques Like Query Rewriting**

* **Query Rewriting helps reformulate user queries into more precise legal terms.**
* **For instance, a user query like "Can a tenant break a lease early?" could be rewritten as "Tenant's legal rights and consequences of early lease termination."**
* **Methods for query rewriting:**
  + **Prompt-based LLM Rewriting: Using an LLM to generate a better-formulated legal query.**
  + **Few-shot learning with legal query examples: Providing multiple variations of how legal professionals phrase similar queries.**

**7. Semantic Chunking**

* **Instead of fixed-size chunking (256 tokens with 64 overlap), semantic chunking ensures that logical sections remain intact.**
* **Legal documents contain structured sections such as "Definitions," "Provisions," "Exemptions," and "Penalties."**
* **Semantic chunking can be achieved using:**
  + **LLM-based document structuring to identify meaningful sections.**
  + **Text similarity techniques (like cosine similarity) to prevent breaking related content apart.**
* **This would lead to better retrieval accuracy and less fragmented legal context during summarization.**

**Challenges:**

**1. Running LLM and Embedding Models Locally**

* **Hosting large models like Llama3 (8B+) and high-quality embedding models locally requires significant computational resources.**
* **Challenges include:**
  + **High GPU/CPU requirements for running inference smoothly.**
  + **Latency issues when processing multiple queries simultaneously.**
  + **Memory constraints when handling large legal datasets.**
* **Potential Solutions:**
  + **Use a hybrid approach: Running smaller models locally and using API-based models for complex tasks.**
  + **Optimize inference by using quantized models (e.g., GGUF for Llama).**

**2. Handling Ambiguous Queries**

* **Legal queries often lack clear context or specificity, making it difficult to route them correctly.**
* **Example: "What are the penalties for fraud?"**
  + **Does it refer to corporate fraud, tax fraud, or financial fraud?**
  + **Without additional details, the system may retrieve irrelevant or overly broad information.**
* **Possible solutions:**
  + **Implement query clarification prompts before retrieval.**
  + **Use multi-turn conversations to refine the user’s intent.**

**3. Parsing Complex Legal PDFs**

* **Legal PDFs contain:**
  + **Multicolumn text, footnotes, tables, and case citations.**
  + **Scanned images that require OCR processing.**
  + **Hyperlinked references that need proper extraction.**
* **Challenges:**
  + **Extracting structured data without breaking logical sections.**
  + **Preserving citation links and formatting.**
* **Solution:**
  + **Use PDF parsers like LayoutLMv3 or document intelligence models to retain structure.**

**4. Finding the Right Chunking Strategy**

* **Fixed-size chunking (e.g., 256 tokens) can cause loss of context if legal provisions get split across chunks.**
* **Semantic chunking is better but requires additional processing.**
* **Challenges include:**
  + **Defining the right chunk size for different document types.**
  + **Balancing overlap to avoid missing context without excessive redundancy.**
* **Possible solution:**
  + **Dynamically adjusting chunk size based on document type (e.g., contracts vs. court rulings).**