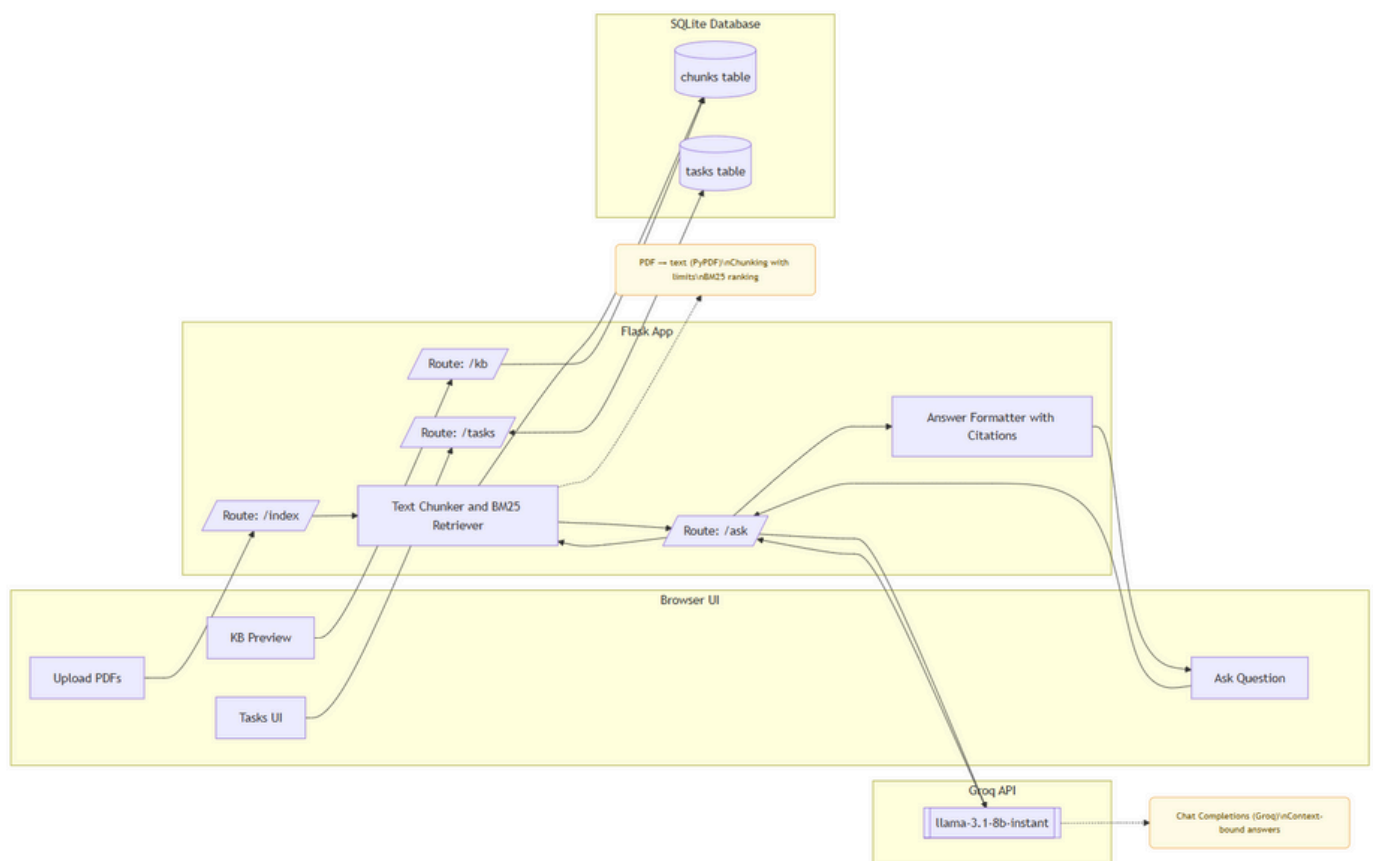


KNOWLEDGE BASE AGENT AND TASK MANAGER

System Architecture

The flowchart provides a high-level overview of the complete system architecture of the AI Knowledge Base and Task Manager Agent. It illustrates how different components—such as the browser interface, Flask backend, SQLite databases, text-processing pipeline, and Groq LLM—work together to deliver document ingestion, retrieval, and task management functionalities. This diagram highlights the end-to-end data flow: from PDF uploads and text chunking, through storage and retrieval, to LLM-powered answer generation and final formatting. By visualizing the interactions between major modules, the flowchart clearly demonstrates how the system processes information and how each part contributes to the overall functionality of the agent.



The flowchart represents the entire **end-to-end workflow** of your AI Knowledge-Base + Task Manager Agent. It shows how the system processes PDF uploads, retrieves information, interacts with the LLM, and manages tasks through the user interface.

1. Client (Browser UI)

The user interacts with the system through a Bootstrap-based web interface. Four main actions exist:

- **Upload PDFs** – send PDF files to the backend for indexing
- **Ask Questions** – submit queries for AI-assisted answers
- **View Tasks** – add, mark, or delete tasks
- **Preview KB** – inspect what text has been stored from PDFs

2. Flask Application (Backend Core)

Your `app_flask.py` file contains all logic for routes:

- **/index** → handles PDF uploads
- **/ask** → processes user questions
- **/tasks** → task creation and listing
- **/kb** → shows stored chunks

Flask acts as the **controller** between UI, database, PDF processing, and the model.

Inside Flask, two important internal components work:

a) CH – Text Chunker & BM25 Retriever

- Extracts text from PDFs using PyPDF
- Breaks text into controlled-size chunks
- Applies BM25-style keyword scoring to rank chunks
- Ensures unique pages are selected
- Prepares the context for LLM

This ensures the LLM only sees **relevant** text.

b) FMT – Answer Formatter

- Takes the raw LLM output
- Converts it into safe bullet points
- Attaches citations (e.g., [source p3])
- Sends clean answer to the UI

3. SQLite Database

The system uses two tables:

- **chunks** – stores extracted PDF text as chunks
 - Fields: source, page, chunk, score
- **tasks** – stores task title, notes, due date, priority, and status

The database enables fast local retrieval and simple task management.

4. Groq LLM API

This component represents the **cloud-based large language model**:

- Receives the question + chunked context
- Applies the system prompt (strict context-only answering)
- Produces citations-friendly answers
- Sends back short, precise text

The model:

llama-3.1-8b-instant (low-latency, accurate for RAG scenarios)

5. Notes (Explanation Nodes)

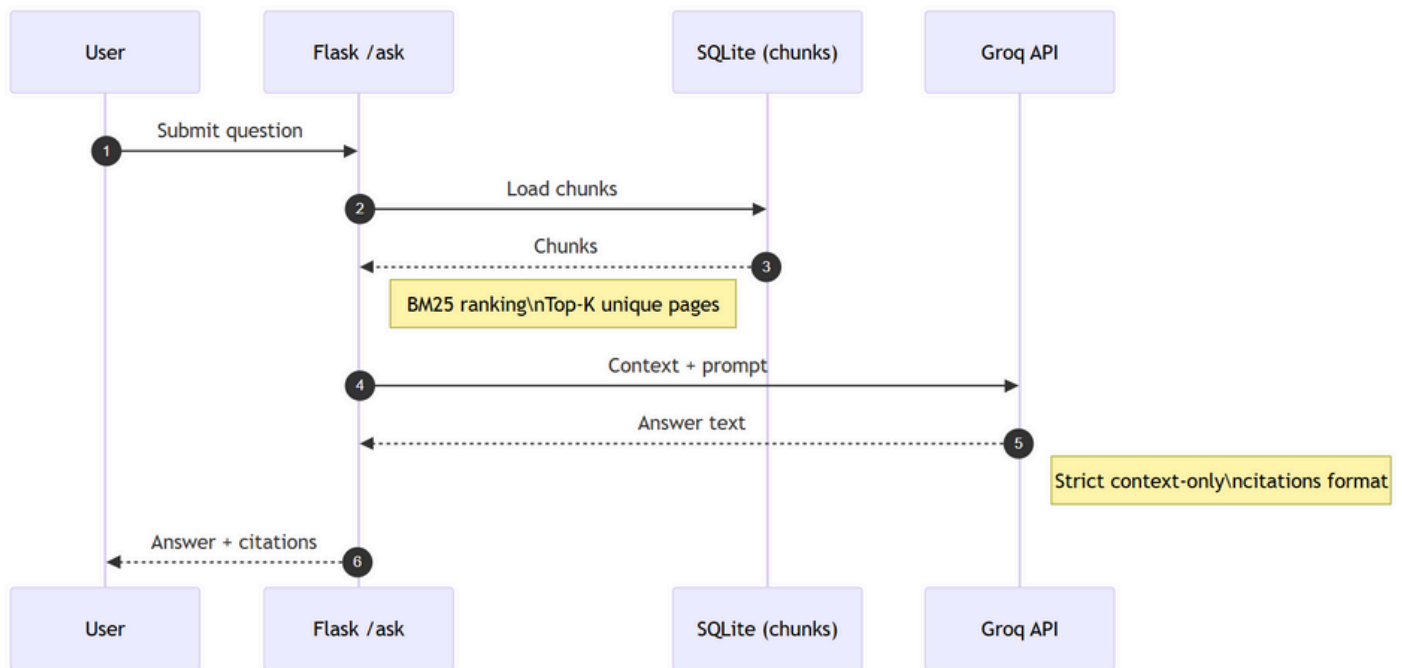
In the diagram, notes describe internal processing:

- PDF → text extraction
- Chunking with overlap
- BM25 scoring
- Context-only LLM answering

These help explain *how* your agent transforms documents into answers.

Sequence Diagram for Knowledge Base Agent

The sequence diagram focuses specifically on the question-answering workflow, presenting a step-by-step view of how a user query travels through the system. It breaks down the communication sequence between the user interface, Flask application, SQLite database, BM25 retrieval logic, and the Groq LLM API. This diagram emphasizes the chronological order in which the system loads text chunks, ranks them for relevance, prepares context, queries the LLM, and formats the final response. By detailing each interaction in the ask cycle, the sequence diagram offers a clear explanation of the internal mechanics that enable accurate, citation-based answers from the uploaded documents.



The sequence diagram focuses on the **Ask workflow only**, showing the exact chronological communication between components.

1. User → Flask (/ask)

The user submits a question through the Ask page. Flask receives the query and optional settings (top-K, model).

2. Flask → SQLite (Retrieve Chunks)

Flask queries the database:

```
SELECT source, page, chunk FROM chunks
```

These are all the stored text segments extracted from PDFs.

SQLite returns the text chunks.

3. BM25 Ranking (inside Flask)

Flask applies BM25 scoring to determine which chunks are most relevant:

- Scores each chunk
- Sorts by descending score
- Selects top-K unique (source, page) pairs

This prevents duplication and produces a **clean context**.

A note in the diagram indicates:

“BM25 ranking, Top-K pages”

4. Flask → Groq LLM (Chat Completions)

Flask sends the following to the LLM:

- The user question
- The selected contextual text
- A system prompt enforcing:
 - Answer ONLY from context
 - Use short citations
 - Compact, bullet-based results

Groq returns the answer text.

A note indicates:

“Strict context-only citations format”

5. Flask → User (Answer + Citations)

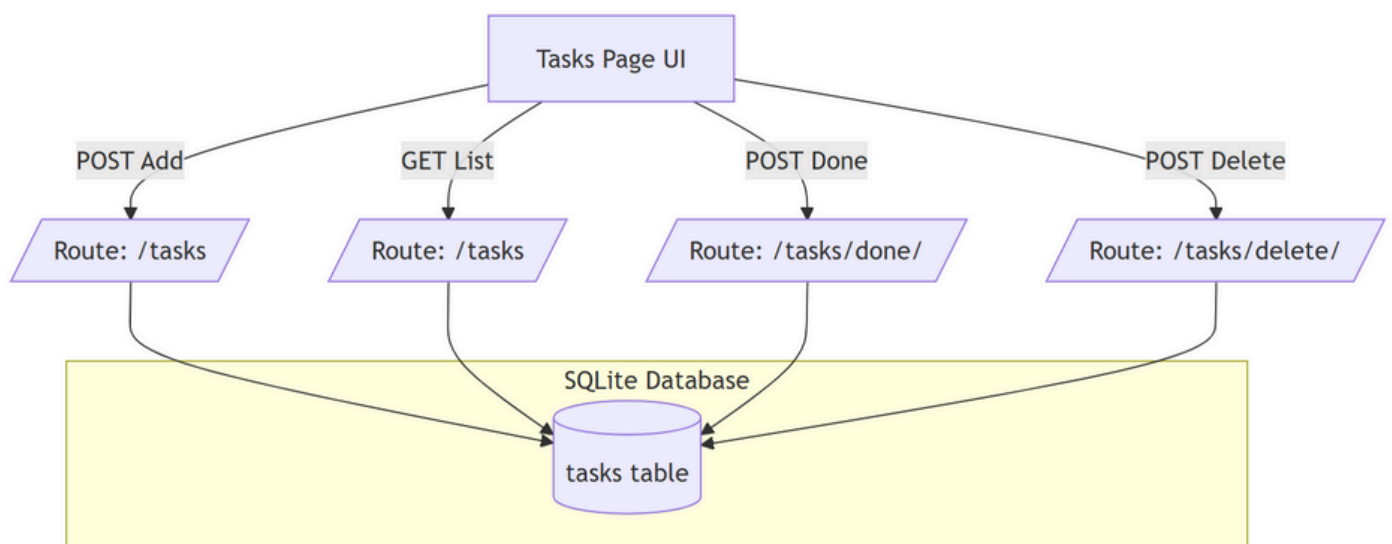
Flask formats the LLM response:

- Converts it to safe HTML
- Adds sources list
- Renders final output in the UI

The user sees:

- Answer
- Bulleted list
- Citations
- Source list

CRUD Operations of Task Manager



The Task Manager CRUD diagram illustrates the complete lifecycle of task operations within the system, showing how the user interface interacts with the Flask backend and the underlying SQLite database to manage task data. It outlines the flow for creating, viewing, updating, completing, and deleting tasks, demonstrating how each user action on the tasks page triggers specific Flask routes that perform corresponding database operations. This diagram provides a concise visual representation of how task-related data moves through the application, ensuring clarity on how the system supports efficient task management alongside the knowledge base functionality.