## Overall Workflow

### 1. Load API Key

* File: config/config.py
* Loads your GROQ\_API\_KEY (or any API key you need).
* Ensures your application can authenticate with external services (LLM provider).

### 2. Process PDF

* File: services/document\_processor.py
* Uses PyPDFLoader to read PDF files into text format.
* Splits large text into manageable chunks with RecursiveCharacterTextSplitter (1000 chars per chunk, 200 overlap).
* Returns a list of document chunks.

### 3. Create Vector Store

* File: services/vector\_store.py
* Takes the chunked documents and converts them into dense vector embeddings using SentenceTransformerEmbeddings (all-MiniLM-L6-v2).
* Stores embeddings in FAISS (Facebook AI Similarity Search), a fast vector database.
* Returns the vectorstore object, which allows semantic search.

### 4. User Query Handling

* Workflow:
  1. User asks a question.
  2. Convert question into embeddings.
  3. Search in vectorstore → retrieve most relevant PDF chunks.
  4. Pass retrieved context + user query to LLM (Groq, HuggingFace, or other).
  5. LLM generates a context-aware answer.

### 5. Fallback to Web Search (Optional)

* If relevant answer is not found in PDF chunks,
  + Use Tavily API / Google Search wrapper to fetch real-time info.
  + Pass result + user query to LLM for final response.

### 6. Final Answer

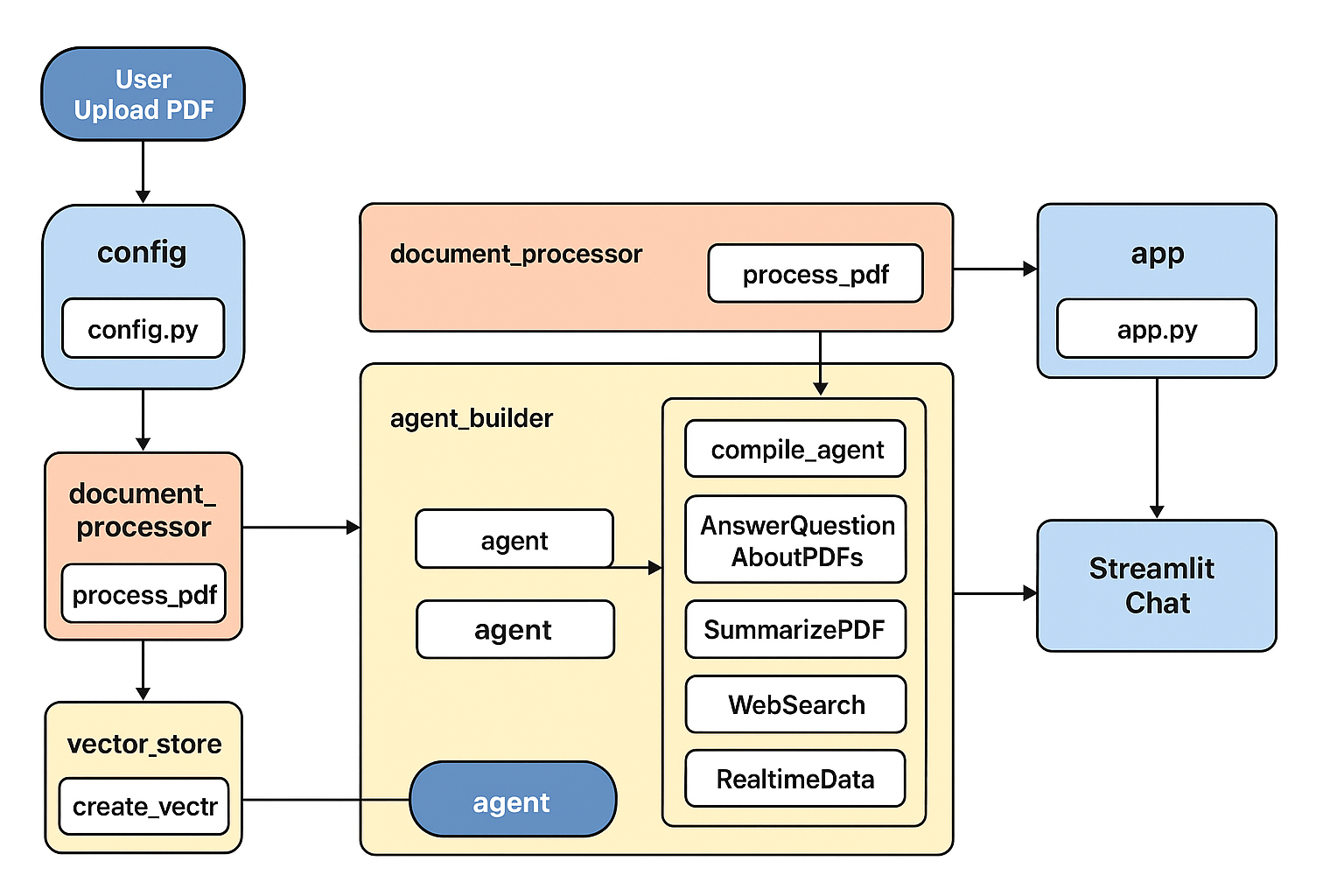
* User gets the best possible answer, either from the PDF knowledge base or from the web.

## Workflow Summary

1. Load API Key → (config/config.py)
2. Upload & Process PDFs → (services/document\_processor.py)
3. Embed & Store Chunks in FAISS → (services/vector\_store.py)
4. User Query → Retrieve Relevant Chunks (semantic search)
5. Pass Chunks + Query to Groq LLM → Final Answer\

## End-to-End Flow (File → Function)

1. User uploads PDF → app.py → process\_pdf(files) → chunked\_docs
2. Convert to vectors → create\_vector\_store(chunked\_docs) → vectorstore
3. Build agent → compile\_agent(vectorstore, chunked\_docs) → agent
4. User asks a question → agent.invoke(input\_data) → agent decides:
   * AnswerQuestionAboutPDFs → RAG tool → LLM answer + confidence
   * SummarizePDF → Summarization tool → summary
   * WebSearch → Tavily search → merge with PDF → LLM
   * RealtimeData → fetch live data
5. Agent returns response → app.py → displays in Streamlit chat



[User Upload PDF] --> main.py:st.file\_uploader()

|

v

[process\_pdf(file)] --> extract text, chunk PDF --> chunks[]

|

v

[create\_vector\_store(chunks)] --> embeddings stored in FAISS

|

v

[compile\_agent(tools, memory)] --> agent with RAG, Web Search, Summarization, Memory

|

v

[User Query] --> main.py:st.text\_input()

|

v

[answer\_query(query)] --> agent.run(query)

|

v

+----------------------------+--------

| PDF Search (RAG Tool) |

| query\_vector\_store(query) |

+----------------------------+--------

| Found Answer? Yes ------> Return PDF Answer

| No

v

+----------------------------+-------

| Web Search (Tavily) |

| tavily\_client.search(query) |

+----------------------------+--------

|

v

Summarization Tool --> Summarize long answer

|

v

[Return Answer] --> Streamlit displays result

|

v

[Memory Updated] --> AgentState stores context