[POC] Architecture and Design of the Auto-CAM-Prototype1: Data Flow and Key Decision

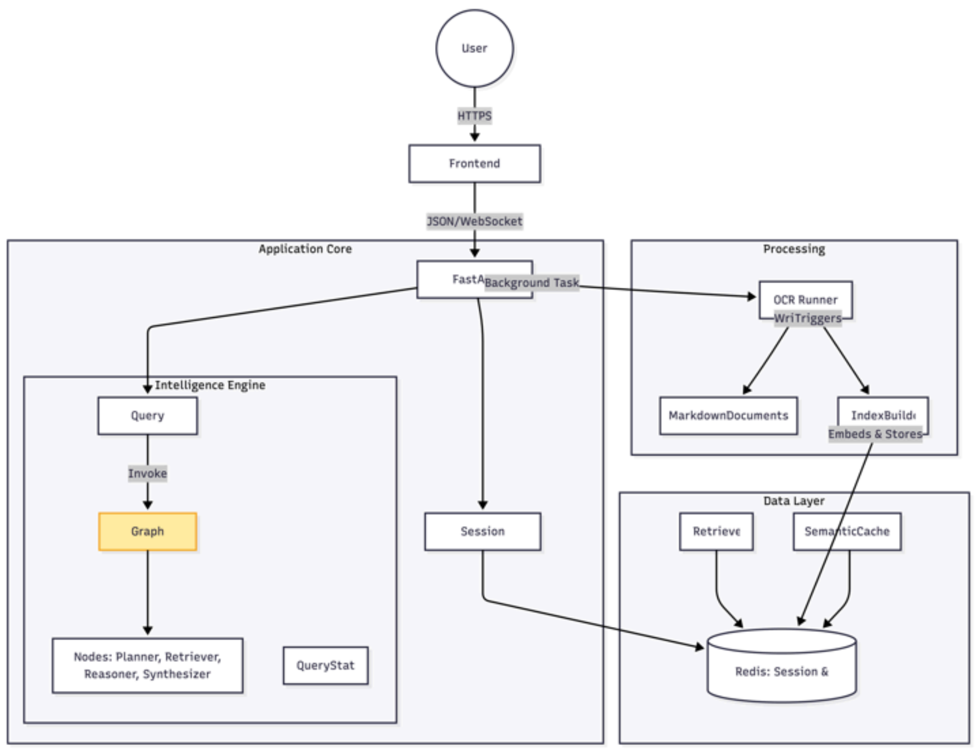


Figure 1: High Level Design

🚀 Detailed Data Flow

1. Ingestion & Digitization (The "Eyes")

The system begins by ingesting raw financial documents (PDFs, Excel).

- **Component**: `OCR Runner`
- **Process**:
  a. **Detection**: Scans directory for files.
  b. **Digitization**: Uses specialized engines (Google DocAI for PDFs, Custom Parsers for Excel) to extract text.

   c. **Normalization**: Converts everything into clean **Markdown** format with structure preservation.
   d. **Output**: `processed_output/*.md` files.

2. Indexing & Storage (The "Library")

We organize text so the AI can find it instantly.

- **Components**:

  IndexBuilder,

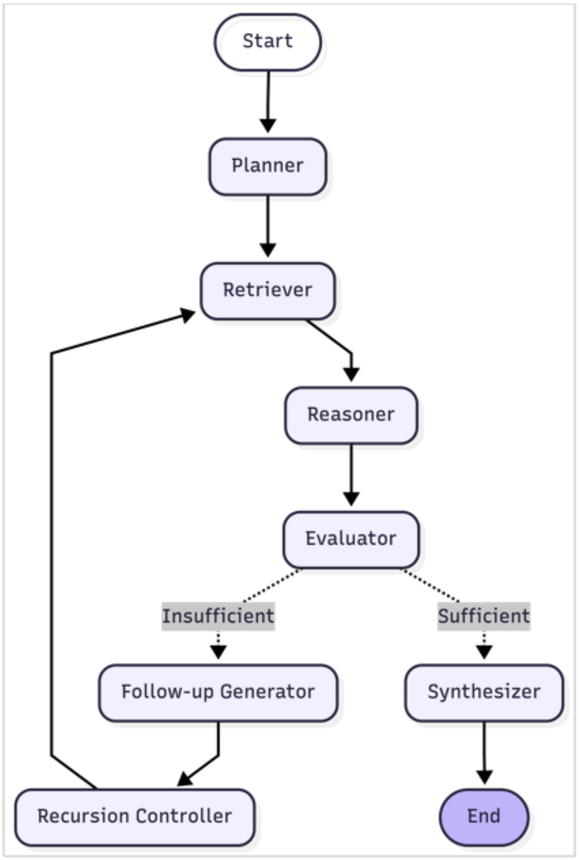
  Chunker, `Embedder`,

  Redis
- **Process**:
  a. **Chunking**:

     MarkdownChunker splits documents into small, logical "chunks" (paragraphs, tables) that fit in the AI's context window.
  b. **Embedding**: Chunks are converted into vectors (lists of numbers) using **TITAN Embeddings v2**.
  c. **Compression & Storage**:
     - **Vectors**: We compress vectors using **Base64 Encoded Numpy Bytes** before storing in Redis.
     - **Technique**: `base64.b64encode(vec_np.tobytes())`.
     - **Advantage**: This is **3x-4x smaller** and faster to transmit than storing vectors as plain text lists (JSON).

3. The Intelligence Engine (The "Brain")

This is the core agentic workflow powered by **LangGraph**.



- **Logic Flow**:
  a. **Planner**: Breaks the user's complex question into specific sub-steps.
  b. **Retriever**: Searches Redis for relevant chunks (Vector Search).
  c. **Reasoner**: Analyzes chunks to answer specific steps.
  d. **Evaluator**: "Did we answer the question?" If no, it triggers **Recursion** (Loop back).
  e. **Synthesizer**: Compiles the final answer.

🧠 Key Design Decisions

Why Redis? (Session Isolation & Speed)

We use Redis as our primary data layer instead of a traditional SQL database.

Advantages:

1. **Session Isolation (Privacy High)**:
   - Every key is prefixed with `session_id`.
   - **Result**: Complete data separation. Session A cannot accidentally touch Session B's data.
   - **Security**: Wiping a session is as simple as deleting all keys with that prefix.
2. **No DB Connections**:
   - Redis is in-memory and extremely fast.

    - We don't manage complex connection pools or schema migrations.
    - This makes the system **stateless** and easy to scale (just add more Redis nodes).

Disadvantages:

- **Cold Start**: Every new session starts from scratch. We must re-index documents for every new "Session ID". (Trade-off for perfect privacy).

High Quality via Recursive Retrieval

Our system delivers exceptionally high-quality answers even from heterogeneous data (long PDFs mixed with Excel).

- **The Secret**: **Recursive Feedback Loops**.
- **How it works**:
    a. The **Planner** breaks a vague question ("Assess the borrower") into specific queries ("Find Father's Name", "Check Credit Score").
    b. The **Evaluator** checks the answer. If the retrieval missed something (e.g., hidden in a footnote), it forces the agent to **try again** with a better search query.
- **Trade-off**: **Time**. This "thinking" process takes longer (multiple LLM calls) but ensures accuracy.

Solving the "Long Context" Problem

LLMs have a limit on how much text they can read (Context Window). Financial documents often exceed this.

- **Solution**: **External Memory (Vector Retrieval)**.
- **Mechanism**: instead of feeding the *entire* 100-page loan agreement to the LLM, we use Redis to find only the **top 5 relevant paragraphs** and feed those.
- **Result**: We can chat with documents of **infinite length** because we only load what is needed for the specific question.