Functional Specification Document (FSD)

Project Name: Visual-First Financial Document Intelligence Agent
Version: 1.0
Date: December 16, 2025
Author: Mukundh Jayapal
Target Role: AI Engineer / Applied ML Intern (Montreal Ecosystem)

1. Executive Summary

The **Visual-First Financial Document Intelligence Agent** is an automated auditing tool designed to process complex financial documents (e.g., 10-K filings, invoices, balance sheets). Unlike traditional OCR solutions that treat documents as plain text, this system utilizes a **Computer Vision-first approach**. It employs a fine-tuned **YOLOv8** object detection model to identify and segment semantic regions (tables, charts, headers) and integrates them into a **Multimodal Retrieval-Augmented Generation (RAG)** pipeline powered by **Google Gemini 1.5 Flash**.

The primary goal is to drastically reduce the time financial analysts spend manually verifying data between unstructured text narratives and structured tabular data.

2. Project Scope

2.1 In-Scope

- **PDF Ingestion:** Mechanism to upload and render multi-page PDF documents.
- **Computer Vision Pipeline:** Detection and cropping of financial tables and charts using YOLOv8.
- **Multimodal Parsing:** "Reading" cropped visual elements using a Vision-Language Model (VLM) to extract structured insights.
- **Knowledge Base Creation:** Indexing textual and visual summaries into a vector database.
- **Chat Interface:** A conversational UI allowing users to query the document (e.g., *"What was the revenue growth in 2024 vs 2023?"*).
- **Deployment:** Hosting the application on a public URL via Streamlit Cloud or Hugging

Face Spaces.

2.2 Out-of-Scope

- **User Authentication:** No login/signup system (MVP is public-facing).
- **Payment Processing:** No monetization features.
- **Batch Processing:** The system handles one document at a time, not batch folders of thousands of PDFs.

---

3. System Architecture

The system follows a **Micro-Service logic** architecture within a monolithic application structure:

1. **Frontend Layer (Streamlit):** Handles user I/O, file uploads, and chat history display.
2. **Vision Layer (Ultralytics YOLO):** Receives PDF pages as images -> Outputs Bounding Box Coordinates -> Crops regions of interest.
3. **Reasoning Layer (Google Gemini + LlamaIndex):**
   - *Ingestion:* Converts cropped images to textual summaries.
   - *Indexing:* Embeds summaries into vector space.
   - *Retrieval:* Fetches relevant context based on user queries.
4. **Storage Layer:** Local file storage for cropped images and a persistent Vector Index (ChromaDB/SimpleVectorStore) for embeddings.

---

4. Functional Requirements (FR)

Module 1: Data Ingestion & Vision Processing

- **FR-01:** The system shall accept PDF files up to 50MB in size.
- **FR-02:** The system shall convert PDF pages to high-resolution images (300 DPI) for processing.
- **FR-03:** The YOLOv8 model shall detect "Table" objects with a confidence threshold > 0.25.
- **FR-04:** Detected tables must be cropped and saved locally with unique filenames linking them to their source page number (e.g., page_5_table_1.png).

Module 2: Multimodal Indexing

- **FR-05:** The system shall pass every cropped table image to **Gemini 1.5 Flash** with a prompt to extract key data points and trends (not just raw OCR).
- **FR-06:** Extracted text summaries must be stored as Document objects in LlamaIndex, containing metadata pointing to the original image path.

- **FR-07:** The system must generate vector embeddings for these summaries using models/text-embedding-004.

Module 3: Chat & Retrieval (RAG)

- **FR-08:** Users shall be able to ask natural language questions regarding the document contents.
- **FR-09:** The system must retrieve the top-k (default k=3) most relevant text/table chunks.
- **FR-10:** The response must cite the source (e.g., *"According to Table 4 on page 12..."*).
- **FR-11:** The system shall maintain chat history context for follow-up questions within the session.

Module 4: User Interface

- **FR-12:** The UI shall display the uploaded PDF in a sidebar or expandable viewer.
- **FR-13:** The UI shall show a status bar indicating the progress of "Detecting Tables" and "Indexing Data."
- **FR-14:** (Optional) The UI shall allow users to view the cropped table images that were detected.

---

5. Non-Functional Requirements (NFR)

- **Performance:**
  - Table detection inference time: < 200ms per page on CPU.
  - RAG Query Latency: < 5 seconds per answer.
- **Reliability:** The system must handle API rate limits (Google Gemini) gracefully by implementing retries or showing user-friendly error messages.
- **Portability:** The codebase must be containerizable via Docker.
- **Security:** API keys must be managed via environment variables (.env or Streamlit Secrets), never hardcoded.

---

6. Technology Stack

Component	**Technology Selection**	**Justification**
Language	Python 3.10+	Standard for AI/ML engineering.
Vision Model	**YOLOv8 (Ultralytics)**	State-of-the-art speed/accuracy for object

		detection.
LLM	**Google Gemini 1.5 Flash**	Native multimodal (Vision+Text), large context window, free tier availability.
Orchestration	**LlamaIndex**	Superior data parsing and RAG abstractions compared to LangChain for this specific use case.
Vector Database	**Local (SimpleVectorStore)**	Zero-latency, no infrastructure setup required for MVP.
Frontend	**Streamlit**	Rapid prototyping, pure Python UI.
Deployment	**Streamlit Community Cloud**	Free hosting, direct GitHub integration.

---

7. Deliverables Checklist

Phase 1: Engineering Core (Week 1-2)

- [x] GitHub Repository created (agentic-rag).
- [x] pyproject.toml dependency management configured.
- [x] YOLOv8 Table Detection pipeline (src/vision/processor.py) operational.
- [x] LlamaIndex Ingestion pipeline (src/rag/ingest.py) operational.

Phase 2: Application Logic (Week 3)

- [ ] app.py developed connecting Vision and RAG layers.
- [ ] Chat interface logic implemented.
- [ ] "View Source Table" feature added (showing the image when cited).

Phase 3: Deployment & Polish (Week 4)

- [ ] requirements.txt frozen for production.
- [ ] Application deployed to Streamlit Cloud.

- [ ] **Demo Video** (Loom/YouTube) recorded (critical for internship applications).
- [ ] **Technical Blog Post** written (Medium/Dev.to) explaining the "Visual-First" architecture.

---

8. Deployment Strategy

Option A: Streamlit Community Cloud (Recommended)

1. Push code to GitHub.
2. Login to share.streamlit.io.
3. Connect repository.
4. Add GOOGLE_API_KEY in the "Secrets" settings of the Streamlit dashboard.
5. Deploy.

Option B: Docker (For "Production" Bonus Points)

- Build a Docker image using the python:3.10-slim base image.
- Install system dependencies (libgl1, poppler-utils) required for OpenCV and PDF processing.
- Expose port 8501.

---

9. Future Roadmap (v2.0)

- **Table-to-Excel:** Allow users to download detected tables as .csv files.
- **Multi-Doc Support:** Compare data across two different PDFs (e.g., Apple 2023 vs Apple 2024 10-K).
- **Local LLM:** Support for running Llama 3.2 locally using Ollama for privacy-centric deployments.