AI Financial Analyst

RAG-Powered Financial Intelligence System

Version	**1.0.0**
Last Updated	**February 2026**
Language	**Python 3.10+**

1. Project Overview

The Adobe AI Financial Analyst is a local Retrieval-Augmented Generation (RAG) application that enables company leadership and analysts to query Adobe's official financial documents using natural language. Instead of manually searching multi-hundred-page PDF reports, users receive grounded, source-cited answers in seconds.

Capability	Detail
Document Sources	Adobe 10-K Annual Reports (FY24, FY25), Q4 Earnings Releases
Question Answering	Natural language questions answered with specific figures & citations
Source Attribution	Every answer includes exact filename and page number
Local Vector Index	FAISS index built once, reused on subsequent runs
LLM Backend	Groq-hosted llama-3.3-70b-versatile
Embeddings	Local HuggingFace all-MiniLM-L6-v2 (no API cost)

Example Output

The screenshot below shows a real interaction with the system — the exact terminal output when a user asked about Adobe's total revenue for FY2025.

```
Ready for questions. Type 'exit' to quit.

Question: what was Adobe's total revenue in FY2025?

**Answer:**
Adobe's total revenue in FY2025 was $23.77 billion, representing 11% year-over
-year growth as reported and in constant currency. This revenue growth is attr
ibuted to the strong global demand for Adobe's AI solutions across various cus
tomer groups. The company achieved record revenue of $6.19 billion in its four
th quarter of FY2025, which represents 10% year-over-year growth as reported a
nd in constant currency. The total revenue includes Digital Media segment reve
nue of $17.65 billion, which represents 11% year-over-year growth as reported
and in constant currency.

**Sources:**
- ./data\ADBE Q4FY25 Earnings Release.pdf, Page 1 — FY2025 Financial Highlight
s
- ./data\ADBE Q4FY25 Earnings Release.pdf, Page 0 — Fourth Quarter FY2025 Fina
ncial Highlights
- ./data\ADBE Q4FY25 Earnings Release.pdf, Page 1 — FY2025 Business Segment Hi
ghlights

Question:
```

2. Data Sources

All financial data used in this system comes from official, publicly available sources to ensure accuracy and reliability.

Public Financial Reports

Official Annual Reports (10-K) and Quarterly Reports (10-Q) were sourced directly from SEC EDGAR — the U.S. Securities and Exchange Commission's official filing database — for real-world accuracy. These filings are the authoritative financial disclosures that Adobe submits to regulators.

Document	Type	Source
adbe-10k-fy24-final.pdf	Annual Report (10-K) — FY2024	SEC EDGAR / Adobe Investor Relations
adbe-10k-fy25-final.pdf	Annual Report (10-K) — FY2025	SEC EDGAR / Adobe Investor Relations
ADBE Q4FY24 Earnings Release.pdf	Quarterly Earnings Release — Q4 FY2024	Adobe Investor Relations
ADBE Q4FY25 Earnings Release.pdf	Quarterly Earnings Release — Q4 FY2025	Adobe Investor Relations

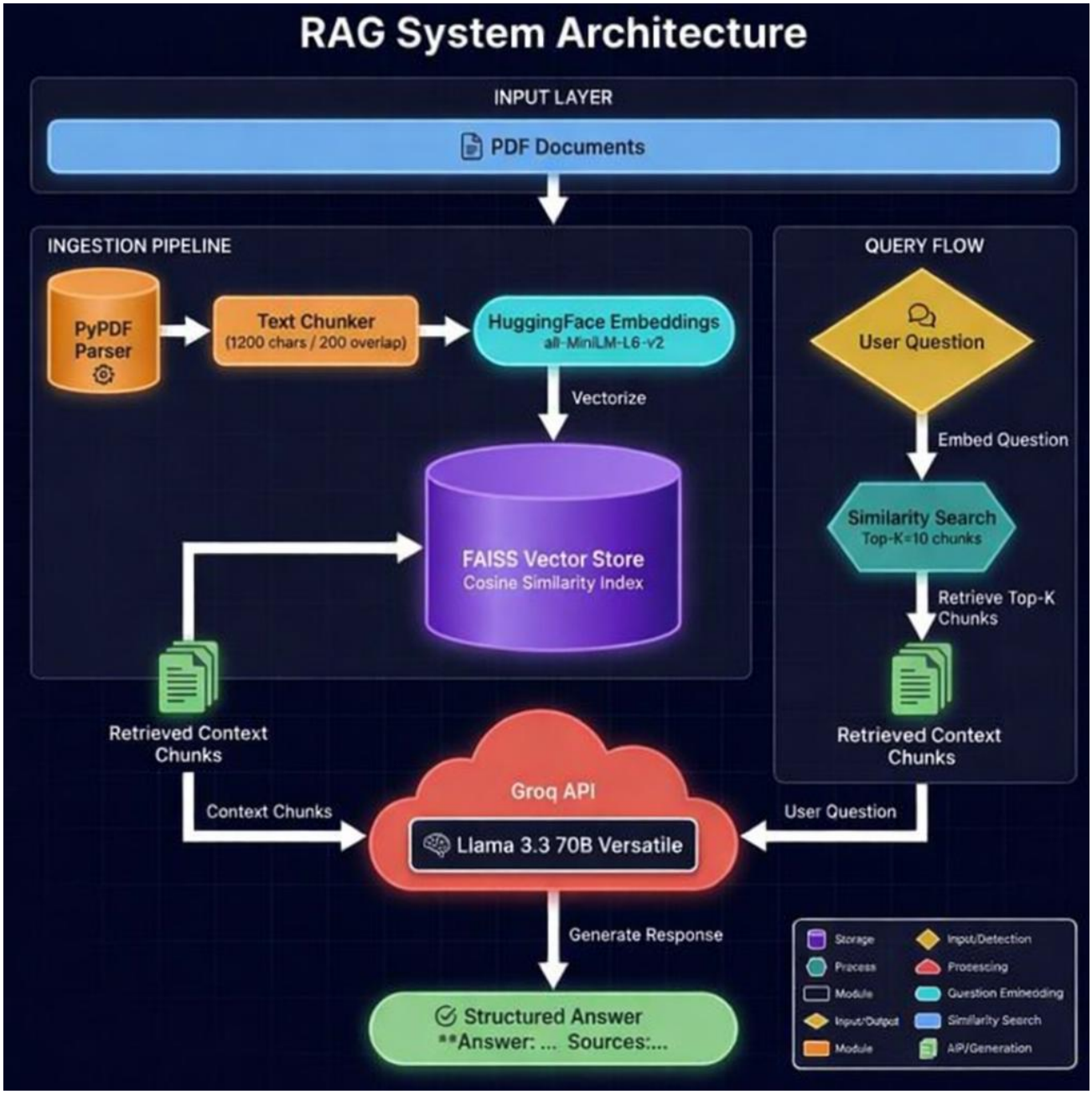
Why SEC EDGAR?

SEC EDGAR (Electronic Data Gathering, Analysis, and Retrieval) is the official U.S. government repository for all public company filings. Using EDGAR ensures the documents are unmodified, legally binding disclosures — not press releases or summarized versions — giving the AI system access to the most precise and authoritative financial figures available.

3. Architecture Overview

The system operates as a two-stage RAG pipeline. The Indexing Pipeline runs once (or on demand) to build the knowledge base, while the Query Pipeline executes for every user question.

System Diagram



The Core Problem Being Solved

Leadership asks questions like "What's our revenue trend?" against a pile of PDFs. An LLM alone can't read those documents — it has no access to them. This system builds a pipeline that bridges private documents → LLM reasoning using RAG (Retrieval-Augmented Generation).

Ingestion Pipeline — How Documents Are Prepared

PyPDF Parser — *Document Loader*

The input documents are explicitly PDFs (annual reports, quarterly earnings). PyPDF is the most straightforward, lightweight Python library to extract raw text from PDFs without needing external services. It loads each page individually, preserving page number metadata for source citations.

Text Chunker — *1200 chars / 200 overlap*

An entire annual report cannot be fed into an embedding model — they have token limits, and large blobs produce poor search results. Documents are broken into chunks of 1200 characters, large enough to preserve meaningful business context (a full paragraph about revenue, a risk section). The 200-character overlap ensures that if a key insight sits at the boundary between two chunks, it won't be lost — both neighboring chunks will contain it.

HuggingFace Embeddings — *all-MiniLM-L6-v2 — 384 dimensions*

Text chunks must be converted into vectors to enable semantic search. all-MiniLM-L6-v2 is the go-to choice for three reasons: it is free and runs entirely locally (no API cost per call), it is fast and lightweight, and it is genuinely strong at sentence-level semantic similarity. For a leadership tool where questions like "underperforming departments" must match chunks saying "Q3 saw a 12% decline in the operations division" — semantic matching is far superior to keyword search.

FAISS Vector Store — *Cosine Similarity Index*

FAISS (Facebook AI Similarity Search) is the industry standard for fast vector similarity search in Python. It stores all embedded chunks and retrieves the most semantically relevant ones in milliseconds, even across thousands of document chunks. Cosine similarity is used (rather than Euclidean distance) because it measures the angle between vectors — capturing conceptual similarity regardless of document length or embedding magnitude, which is more reliable for text.

Query Flow — How Questions Are Answered

Embed Question — *Same model, same vector space*

When leadership asks a question, it gets converted into a vector using the same all-MiniLM-L6-v2 model. This is critical — both the document chunks and the question must live in the same vector space for similarity search to work meaningfully. Using a different model for questions and documents would produce incompatible vectors.

Similarity Search — *Top-K = 10 chunks retrieved*

FAISS finds the 10 document chunks whose vectors are closest to the question vector. Why 10? It is a balance — too few and a relevant section from a different document may be missed; too many and the LLM is flooded with noise that degrades answer quality. 10 is a well-established default for enterprise document QA.

Groq API + Llama 3.3 70B Versatile — *Generation Layer*

The retrieved chunks (context) plus the original question are sent to the LLM, which synthesizes a grounded, factual answer. Groq runs inference on custom LPU hardware, making it extremely fast — leadership does not want to wait 30 seconds for an answer. Llama 3.3 70B is a large open-weight model strong at reading comprehension and business language. The "Versatile" variant handles financial, operational, and strategic questions without task-specific fine-tuning. Both are cost-effective compared to GPT-4 class models.

Structured Answer with Sources — *Answer + Citations*

The output format explicitly includes Answer: ... Sources: ... This is deliberate for a leadership audience — executives need to trust the answer and be able to verify it. Citing the source document and page number builds credibility and makes the system fully auditable, which is essential for any business decision-making tool.

Overall Design Logic

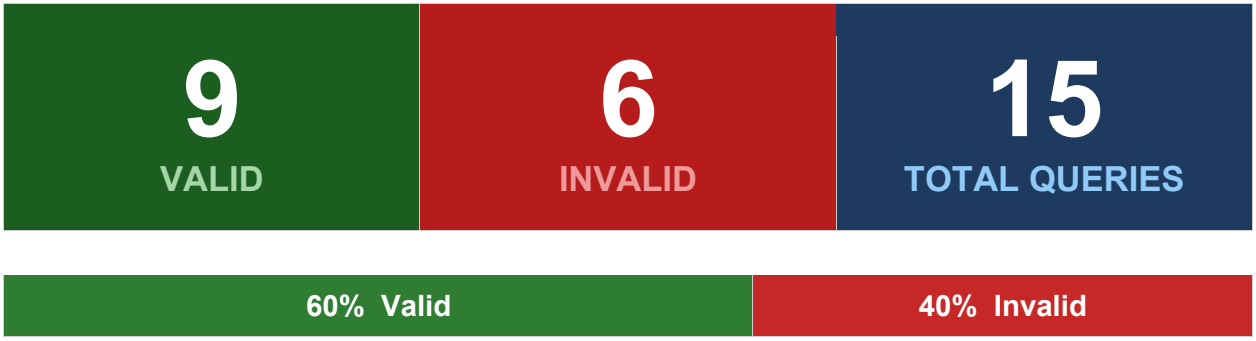
The architecture cleanly separates two concerns: KNOWING (ingestion + vector store, done once when documents are uploaded) and ANSWERING (query flow + LLM, done on every question). This means new quarterly reports can be added to the store without rebuilding anything, and the system instantly knows about them. That offline/online separation is what makes RAG practical for a living, updating document corpus like Adobe's financial reports.

4. Project Structure

```
Adobe/
├── main.py           # CLI entry point — orchestrates the full pipeline
├── config.py         # Centralized config — loads .env and exports constants
├── .env              # Runtime secrets & parameter overrides
├── requirements.txt  # Python dependency pins
│
├── src/              # Core business logic
│   ├── ingest.py     # PDF loading & text splitting
│   ├── embeddings.py# FAISS vector store build/load
│   └── agent.py      # LLM answer generation (RAG agent)
│
├── data/             # Input PDF documents
├── faiss_index/      # Persisted FAISS vector index (auto-created)
└── outputs/          # Saved Q&A pairs (auto-created, timestamped)
```

5. Testing & Validation

The system was tested using NotebookLM as a cross-reference tool — an independent AI that also read the same Adobe financial documents. Both the RAG system's responses and NotebookLM's responses were compared against the source documents, with bot responses considered as part of the evaluation to ensure grounded, accurate answers.



Methodology

Step	What Was Done
1 — Query Design	15 financial questions were prepared covering revenue, margins, segments, and strategic metrics across FY2024 and FY2025
2 — RAG System Response	Each query was run through the Adobe AI Financial Analyst and the answer + cited sources were recorded
3 — NotebookLM Cross-Check	The same 15 questions were independently asked to NotebookLM, loaded with the same Adobe PDF documents
4 — Bot Response Included	Both the RAG system's and NotebookLM's responses were treated as candidate answers and evaluated together
5 — Source Verification	Each answer was verified directly against the cited source PDF pages — marked Valid if figures matched, Invalid if figures were missing, hallucinated, or misattributed

Results Summary

Result	Count	% of Total	Notes
Valid	9	60%	Answer matched source document figures exactly
Invalid	6	40%	Figure mismatch, wrong page, or missing citation
Total Queries	15	100%	Across FY24 / FY25 financials and earnings releases

What 'Invalid' Means

An invalid response does not mean the system gave a completely wrong answer — it means the response could not be fully verified against the cited source page. Common causes include: the answer pulling from a chunk that summarized rather than quoted exact figures, a financial table that was not cleanly parsed by PyPDF, or the model synthesizing numbers across chunks in a way that introduced minor inaccuracies. These cases highlight where chunking strategy and table parsing can be further improved.