Presentation Deck

Outlines

- Overview and our target user
- Key product decisions
- Technical challenges encountered
- Success Metrics and Tracking Strategies
- Next iteration priorities

Overview

Our Jupyter-based RAG system helps financial analysts extracting and synthesizing key insights from financial reports

Background

Our Jupyter-based RAG system helps insurance analysts quickly extract insights from 10-K reports, streamlining competitive research and strategic planning.

Target user profile

Our target users are **insurance industry financial analysts** specializing in competitive intelligence.

They navigate 10-K reports using structured frameworks, prioritizing **accuracy and speed**.

Our RAG system must deliver efficient, accurate insights with minimal friction.

Target user flow and focus area

Step 1: Document Collection
Step 2: Information Extraction
Step 3: Analysis
Step 4: Insight Synthesis
Step 5: Report Formatting

We'll alleviate the pain of step 3 to step 4 through our RAG system. However, step 2 is included in the scope because identifying the accurate key information is critical in achieving the following steps.

Key Product Decisions

Once the key problem areas are identified, the best technical alternatives based on Jupyter Notebook on local machine setup are chosen

Decision 1: Prioritizing Problem

Step 1: Document Collection
Step 2: Information Extraction
Step 3: Analysis
Step 4: Insight Synthesis
Step 5: Report Formatting

Steps 1 (Document Collection) and 5 (Report Formatting) are excluded from the current iteration, assuming analysts already have standardized data sources.

While report formatting is cumbersome, it is not a critical part of the value chain for delivering key insights.

Decision 2: Model Architecture

GPT-4 API vs specialized models for different stages (e.g. querying, chunking, indexing, embedding)

Considering the Jupyter Notebook interface, while specialized models could optimize individual components and reduce costs, the unified approach simplifies development and maintenance.

The consistency and reliability of GPT-4 outweigh the additional API costs for this financial analysis PoC.

Decision 3: Chunking Strategy

Fixed size token, semantic chunking, vs **hybrid approach**

A hybrid approach is implemented instead of pure fixed-size or semantic chunking.

While pure semantic chunking would offer better context preservation, the hybrid approach reduces complexity while maintaining accuracy for financial analysis.

Decision 4: Vector Strategy

Pinecone, Weaviate, vs FAISS

Local FAISS implementation with L2 distance metrics was chosen over cloud solutions like Pinecone or Weaviate.

Though cloud solutions offer better scalability, FAISS provides adequate functionality without external dependencies.

Decision 5: Context Management

In-memory Python dictionary, permanent storage (SQL, NoSQL, Vector)

For a PoC RAG system, an in-memory Python dictionary enables fast, simple context tracking without the complexity of database setup. While it lacks long-term persistence, it efficiently maintains 3-4 recent interactions, making it a practical choice for iterative development.

Technical Challenges

Choosing the best notebook setup, surprisingly, took the most time out of my RAG system development

Choosing the notebook interface with least interruption

- Google Collab
- AWS SageMaker Notebook
- Azure Notebooks
- Local iPython (Jupyter Notebook)

I chose a local Jupyter Notebook for stability and efficiency. Google Colab's frequent crashes disrupt workflows, while AWS SageMaker and Azure Notebooks require setup time I aimed to avoid. A local setup enables faster iteration, direct resource control, and minimal overhead.

rag.ask() function only returns one company

I tried multiple times to make sure that every company in comparison are mentioned in the answer.

The issue arises because the current implementation doesn't properly track which chunks in the FAISS index belong to each company. When retrieving relevant chunks, it incorrectly matches global FAISS indices to per-company chunk indices.

Run time error

I frequently encounter runtime errors and have to restart the kernel multiple times.

When processing PDF files, I often run into errors.

To avoid manually splitting PDFs into sections—which would add complexity by requiring me to manage scattered files for the same company—I improved the code's memory management.

Key Metrics

The key success metrics span across the 3 key objects of accuracy, efficiency (speed & token cost), and user satisfaction

Objective	**Accurate Financial Fact Extraction**	**Efficient System Performance**	**Enhance User Experience**
Key Goals	• **Achieve 95% accuracy in financial fact extraction** • **Maintain 95% relevance score**	• **Keep 95% of Queries under 5-Second response time** • **Maintain around 5K token per request (~$0.08/req)**	• **Retention: 90% of inquiries use this RAG system** • **SCAT: 85%+ of users find the system's reliable and actionable**
Metrics	**Relevance Score** • Measures if retrieved chunks directly answer the query • This can be implemented via post-answer user feedback buttons • If repeated negative feedback is registered on certain carriers, investigate chunk alignment or embedding drift **Cross-Company (insurers) Recall** • Tracks if query asks for multiple entities, the response covers each entity evenly (as opposed to uneven) • This can be implemented via Named Entity Recognition (NER) in the code and human validation **Hallucination Rate** • Tracks hallucination • This can be implemented via LLM self-check code and human audit	**Latency (each for Retrieval, Augmentation, Generation)** • Measures time to search, process, and LLM response • Log timestamps before/after FAISS search for retrieval • Time chunk processing and prompt engineering for augmentation • LLM API response time tracking for generation **Token Usage (each for Retrieval, Augmentation, and Generation)** • Tokens consumed to embed query + retrieve chunks for Retrieval • Tokens used to format context for Augmentation • LLM input/output tokens for Generation which can be calculated based on API usage report	**Daily Use Adoption Rate** • Measures % of target users actively engaging with the system daily • Track unique daily users via login/API keys **Overall Satisfaction (SCAT)** • User-reported satisfaction with system outputs • Anecdotal by manually inquiring users **Trust in Insights** • Confidence that system outputs are reliable/actionable • Anecdotal by manually inquiring users **Task Completion Rate (Perceived)** • % of users who believe the system helped them finish tasks • Anecdotal by manually inquiring users

Compliance Note: PII-related metrics are excluded as the system processes only publicly available financial documents, which by regulatory disclosure standards do not contain sensitive personal or non-public information.

Next interaction will focus on increasing the trust on numerical figures presented by RAG followed by enhancing multi-company analysis, conversation memory

Values Achieved from Current Iteration

Analysts see numbers directly from source documents with page references

Responses group related metrics (e.g., “Loss Ratios” + explanatory text)

Follow-up questions retain company names and prior topics

Analysts can refer to prior answers within the same chat

Next Iteration

Pain Point Addressed	Planned Improvement	User Benefit
I need to trust these numbers match the source	Add automated number validation against PDF tables	Red highlight shows mismatches
Why didn’t it mention Chubb?	Guarantee responses reference ≥2 insurers	Answers always include *“Compared to [Company Y] on page…”*
Where’s the full context for this table?	Expand table-text linking to 5-line context	Click any number to see *full table + surrounding analysis*