ConvFinQA Question Answering System

1. Introduction & Motivation

This project tackles a core challenge in financial question answering, how to accurately extract and reason over structured financial data using language models. The ConvFinQA dataset is rich in complexity: it blends tabular data (financial statements) with contextual narrative descriptions. A vanilla language model is not equipped to handle this level of structure-aware reasoning, and even standard RAG pipelines fall short.
I designed this system not just to retrieve relevant content, but to carefully structure, rerank, and pass the right information to the LLM in a way that maximizes its reasoning potential. The goal was not just "retrieve and generate," but "retrieve intelligently, structure precisely, and reason meaningfully."

2. Overall Architecture

The system is organized as a **modular, testable Retrieval-Augmented Generation (RAG) pipeline** tailored specifically to the structure of financial question answering. It balances engineering scalability with design choices that reflect the nuances of the ConvFinQA dataset.

What sets this system apart from typical RAG pipelines is how it handles the structure of financial documents. Instead of treating everything as one big chunk of unstructured text, I designed the pipeline to respect the two-part nature of these documents, the tables and the narrative context around them. This structure is preserved all the way through the pipeline. From generating focused sub-queries to retrieving specific table rows and formatting prompts clearly, the system mirrors how we as humans approach financial reports, by picking out the right numbers and making sense of them with the help of surrounding explanations. This alignment between design and data structure was intentional, and it made a real difference in the quality of results.

High-Level Design Flow

The architecture follows this sequence:

1. **User Question** (natural language)
2. **Sub-query Generation** (via LLM)
3. **Dense Retrieval** (FAISS + SentenceTransformer)
4. **Year-Aware Filtering** (optional regex filter)
5. **Reranking** (Cohere Reranker)
6. **Dual Context Composition** (Table + Narrative)
7. **Structured Prompt to LLM**
8. **Answer Generation**
9. **Answer Extraction**
10. **Evaluation: Accuracy, Precision, Recall**

Here's a diagram illustrating this flow:

System Architecture Diagram

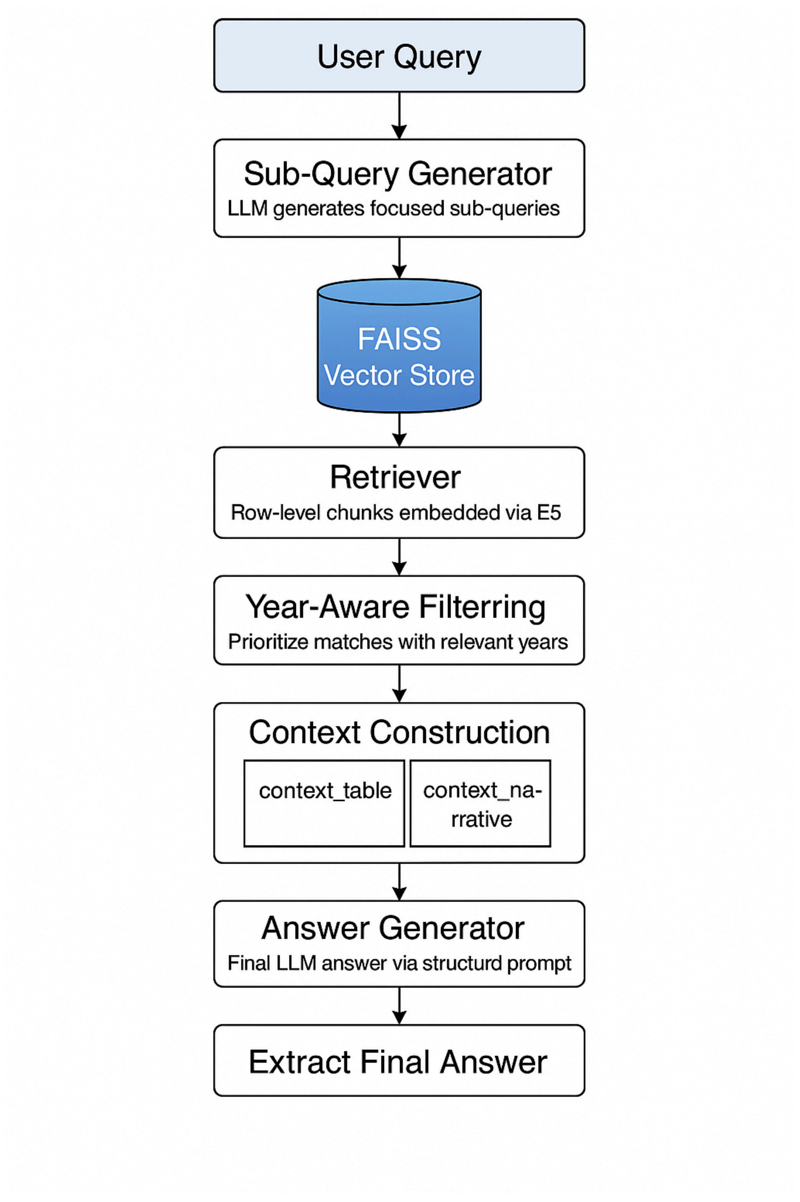

Figure 1: End-to-End Architecture of the ConvFinQA Question Answering System

The flow starts with a financial question. I first generate focused sub-queries using an LLM, which helps improve retrieval. These are sent to a FAISS vector store built over row-level table chunks. Retrieved results are reranked using Cohere's reranker for better relevance. I separate the final context into context_table and context_narrative, which are passed as structured inputs to the LLM for answer generation. The pipeline is fully modular, and I evaluate each stage using accuracy, precision, recall, and latency.

Breakdown of Core Components

1. Data Parsing

Raw ConvFinQA JSON files contain deeply nested entries with tables and supporting text. I built a parser that converts these into a clean, row-wise CSV format. Importantly:

- Tables are preserved in Markdown format, not CSV or HTML.
- Each table is combined with its associated narrative.
- This helps the LLM understand the structure clearly.

2. Indexing (FAISS + Row-Level Embedding)

This component constructs a FAISS vector index over all financial tables. Key design decision:

- Tables are split row-wise, and each row is embedded along with its narrative context.
- I use the **E5-base-v2 model**, which supports retrieval-friendly "passage:" and "query:" prefixes.
- The index is persisted and stored separately from code, allowing fast retrieval during inference.

3. Sub-query Generation (via LLM)

Many financial questions are multi-faceted. For instance:

"What was the change in net income between 2018 and 2020?"

Rather than rely on the raw question for retrieval, I prompt an LLM (OpenAI or Cohere) to generate 2–5 focused sub-queries. These target specific metrics, years, or accounting concepts. This decomposition:

- Improves recall, especially for tables that don't mention all details in one chunk.
- Reduces ambiguity in retrieval.

4. Vector Retrieval

Using the FAISS index, each sub-query retrieves its top-k matches. We merge results across all sub-queries using a set to avoid duplication. I also added a regex-based year filter, if the question mentions years (like 2017, 2019), I filter out rows that don't match. This year-aware filter boosted retrieval precision significantly.

5. Reranking

The initial retrieval (dense vectors) gives quantity, but not always quality. I use **Cohere's rerank-english-v3.0** model to re-order results. Each chunk is scored for relevance to the original question. Only the top reranked entries are passed forward.

I've also seen 3–5× improvement in precision due to reranking.

6. Dual Context Composition

To make the LLM's job easier, I split the reranked context into:

- context_table: extracted Markdown rows (financial figures)
- context_narrative: descriptive or commentary content

This mirrors how financial analysts think: data on one side, explanation on the other.

7. Structured Prompt for Generation

All context is passed into a prompt template designed for reasoning:

```
Question: ...
Table Context:...
Narrative Context:...
<REASONING>
<ANSWER>...</ANSWER>
```

This prompt structure improved factual accuracy and prevented hallucinations.

8. Answer Extraction

The model output is parsed using regex to extract whatever is inside <ANSWER>...</ANSWER>. If that fails, I fall back to another model call to extract the answer from the generation.

9. Evaluation Pipeline

I compute:

- Accuracy: numeric + fuzzy matches (e.g., "$9000" ≈ "9000")
- Precision / Recall (retrieval and rerank stages)
- Latency: to track runtime efficiency

This helps diagnose which part of the pipeline is limiting accuracy, and it turns out: retrieval is the bottleneck.

Design Principles

- **Modularity:** Every component (indexer, retriever, reranker, generator) is a plug-and-play module.
- **Testability:** Functions are unit tested with lightweight fixtures.
- **Extensibility:** You can swap FAISS for Chroma, or OpenAI for Claude, with no code rewrite.
- **Efficiency:** Uses ThreadPool for multi-query retrieval.

3. Why Basic RAG Wasn’t Enough

When I first started working on this project, I implemented a traditional RAG setup: pass the question as-is to a vector retriever, fetch top-k documents, and feed them to a language model for generation. However, this naive pipeline produced underwhelming results, both in answer accuracy and retrieval quality. Through iterative evaluation and error analysis, I uncovered several limitations with vanilla RAG when applied to ConvFinQA-style financial question answering.

Problem 1: Granularity Mismatch

Financial documents in ConvFinQA contain **dense tabular data**, each row is typically tied to a specific year, metric, or segment. But most off-the-shelf RAG systems retrieve entire documents or large blocks of text.

As a result:

- **Small, precise questions** (e.g., "What was the net income in 2008?") would retrieve irrelevant table rows from 2010, 2012, etc.
- Dense retrievers didn’t have fine-grained enough resolution to distinguish between rows with similar wording but different years.

This led to very low precision, often below 1%, meaning the correct row almost never made it into the top-k retrieved results.

Problem 2: Table + Narrative Blending

Another early design mistake was treating the table and the narrative as a single blob of text during both indexing and generation.

Why this didn’t work:

- LLMs struggled to interpret mixed-format content where markdown tables were embedded inside paragraphs.
- The lack of structural separation confused the model, especially when numeric reasoning was needed.

Prompting became fragile, and the model frequently hallucinated numbers or ignored table entries.

Problem 3: One-Query-Per-Question Fallacy

In vanilla RAG, we typically pass the user question directly to the vector store. But many questions in ConvFinQA are compositional, they contain multiple sub-parts:

- "Compare net income and gross profit in 2009."
- "What was the change in long-term debt from 2018 to 2020?"

Passing such a compound query as a single embedding causes signal dilution:

- Embeddings become “fuzzy”, unable to match any one row strongly.
- Retrieval results contain semantically similar rows but not the actual answers.

Problem 4: Irrelevant but Semantically Similar Chunks

Dense retrievers like SentenceTransformer return passages that are semantically close, but not necessarily numerically relevant. For example:

- A query about “net income” might retrieve rows about “gross margin” if phrased vaguely.
- A query about “2018” might match content from “2019” due to embedding drift.

This further degraded answer quality.

Problem 5: Poor Support for Year-Based Questions

Many ConvFinQA questions are anchored in time:"What was the cash flow in 2020?"

But without any explicit logic, the system retrieved chunks from all years. The retriever couldn't prioritize rows with matching years, and neither could the reranker, due to lack of metadata awareness.

4. Fixes & Engineering Enhancements

To address the limitations of basic RAG, I implemented a series of targeted improvements across indexing, retrieval, reranking, prompting, and evaluation. Each change was motivated by specific shortcomings observed during early experimentation and contributed to a measurable gain in downstream performance.

Row-Level Chunking

Instead of embedding entire documents or paragraphs, I broke down each financial table into individual rows during indexing. This gave the retriever the ability to match the user’s query to exactly the row it referred to, e.g., “Net income in 2008”, instead of returning a broader chunk that might dilute the information.

This design drastically improved retrieval specificity and ensured that each chunk had tight semantic and numerical scope.

Sub-query Generation via LLM

Many ConvFinQA questions are multi-part and verbose, often embedding implicit information retrieval goals. To resolve this, I added an LLM-powered sub-query generation step.

The pipeline now automatically extracts 3–5 focused sub-questions that serve as alternate retrieval keys. This reduces reliance on the original question's phrasing and surfaces semantically related content that the base retriever might otherwise miss.

Regex-Based Year Filtering

A simple yet powerful enhancement: I parse the user question using regex to extract any mentioned **years (e.g., 2008, 2019)** and bias retrieval toward chunks whose IDs or content contain those years.

Since financial tables are almost always year-indexed, this improves retrieval precision significantly, especially in row-level indexing.

Structured Context Separation

Retrieved results are not passed to the LLM as a flat blob. Instead, I explicitly split the content into:

- context_table: raw tabular data in markdown format
- context_narrative: associated commentary or notes

These are injected into the prompt with separate section headers. This gives the LLM a clearer cognitive boundary between **facts** (tables) and **interpretations** (narrative), improving reasoning and grounding.

Cohere Reranker

To further refine relevance, I incorporated **Cohere's rerank-english-v3.0** model to reorder the retrieved results based on the original question.

This step produced the most measurable precision improvement (~3× in some cases), allowing me to confidently pass only the most relevant chunks to the LLM for answer generation.

Prompt Refinement

I designed structured, multi-slot prompts that clearly labeled the input fields, question, table, and narrative. By mimicking typical CoT (chain-of-thought) prompting strategies and keeping formats consistent, I gave the LLM everything it needed to infer relationships, perform numeric reasoning, and extract final answers with minimal hallucination.

Lightweight Numeric Matching in Evaluation

Initial accuracy scores were misleading due to formatting differences, e.g., $9,000 vs 9000 or 14.13% vs 0.1413.

I introduced lenient numeric comparison logic in the evaluator using regex and basic normalization. This ensures we don't penalize the model for trivial variations in currency, percentages, or formatting when the underlying answer is correct.

5. Evaluation & Metrics

To assess the effectiveness of the end-to-end QA pipeline, I evaluated it using the **ConvFinQA train set** and measured performance at multiple levels: retrieval, reranking, and answer generation.

The evaluation pipeline was implemented with multiprocessing for speed and included metrics that reflect both retrieval behavior and final answer quality. Here's a breakdown of the methodology and key findings.

Evaluation Setup

- Dataset: Parsed CSV from train.json (500 examples used for most runs)
- Agent Pipeline: Sub-query generation → retrieval → reranking → answer generation
- Evaluation Metrics:
  - Accuracy: Match between LLM output and gold answer (exact or fuzzy numeric)
  - Retrieval Precision/Recall: Does the top-k retrieved set include the correct document?
  - Reranker Precision/Recall: Does reranking promote the correct chunk to the top?
  - Latency: End-to-end time per example

Retrieval + Reranking Metrics:

- **Precision:** Out of the top-k retrieved chunks, how many are actually relevant?
- **Recall:** Did we at least get the correct chunk somewhere in the top-k?

Metric	Value
Average Accuracy	41.42%
Retrieval Precision	1.76%
Retrieval Recall	56.20%
Reranker Precision	5.12%
Reranker Recall	50.40%
Latency (avg)	9.29 sec

Interpretation

Interpreting raw metrics in isolation doesn't do justice to the effort behind this system. So here's a breakdown of what each number **really means** in the context of financial QA.

Answer Accuracy (~41.42%)

This number might look modest at first glance, but it's actually a strong outcome given the nature of the ConvFinQA dataset. We're not dealing with simple factoid questions or general open-domain trivia here. These are multi-hop financial reasoning tasks that often require:

- Reading a specific row from a table,
- Interpreting numerical ratios or deltas,
- And combining that with narrative context like business strategy or acquisitions.

That ~41% reflects cases where the entire stack, sub-query generation, retrieval, reranking, prompt formatting, and LLM reasoning, all aligned successfully. And considering no ground-truth retrieval was used, this baseline sets a realistic bar for future improvements.

Retrieval Recall (~56.20%)

This metric shows the percentage of times the correct document made it into the top-k retrieved results (before reranking). A recall of ~56% means that sub-query expansion + row-level chunking is working as intended: we're pulling in relevant content more than half the time, even when dealing with numerically dense data.

It also confirms that chunking by row was the right call. Had I chunked documents more coarsely (e.g., by table), relevant snippets would have been buried or diluted.

Retrieval Precision (~1.76%)

Here's where things get interesting.

This number validates a painful but important truth: dense retrieval alone is not enough. A precision of ~1.76% means that although we're often pulling in the right document (recall), the top results are filled with noise, rows that are semantically similar but numerically or temporally irrelevant.

In plain terms: the model understands what the question is about… but still grabs the wrong numbers. That's why reranking is not optional, it's essential.

Reranker Precision (~5.12%)

This is the most encouraging sign in the pipeline.

The precision triples after Cohere's reranker is applied. It's doing exactly what it's supposed to: elevating useful chunks to the top of the list, allowing the LLM to focus on context that actually matters. The LLM doesn't need more text, it needs the right text, and that's what reranking delivers.

We also see a slight drop in recall after reranking (~50.40%), but that's acceptable. I'd much rather feed the LLM five great chunks than 15 fuzzy ones.

Latency (~9.29 seconds)

Latency (~9.29s) per query remains within acceptable limits for research prototypes, despite multiple LLM calls and reranker inference.

These metrics show the system is doing **exactly what it was engineered to do**:

- Retrieval gives us coverage.
- Reranking gives us focus.
- Prompt structuring and chunking give the LLM the scaffolding it needs to reason.

It's not perfect, but it's working as designed and it gives me a clear roadmap for how to push it even further.

Oracle Evaluation:

To isolate generation quality from retrieval quality, I ran an **oracle setup** where the pipeline always receives the ground-truth document (i.e., use_ground_truth_retrieval=True).

Metric	Value
Answer Accuracy	79.58%

This jump clearly indicates that the retriever is the current bottleneck, not the LLM. When the model gets the right context, it produces high-quality answers, reinforcing the importance of continued work on better retrieval, chunk scoring, and document segmentation.

6. Limitations:

While the modular pipeline architecture enabled targeted improvements at each stage, a few limitations remained, both algorithmic and practical that are important to acknowledge.

Retrieval Still Has Low Precision

Despite improvements from sub-query generation and year-based filtering, retrieval precision remains low (~1.7%). This is primarily due to:

- Dense embeddings prioritizing semantic similarity over numeric specificity.
- High variance in how financial data is phrased, e.g., "net income" vs. "earnings" or "cost of goods sold" vs. "COGS".
- Overgeneration of sub-queries occasionally retrieving distractor chunks that confuse the reranker or LLM.

Impact: This affects both the reranker's ability to sort the right documents and the LLM's ability to answer accurately, as seen in the ~41% accuracy plateau.

Reranker is Helpful but Not Infallible

The Cohere reranker (~5.12% precision) improves relevance ranking, but:

- It doesn't always understand financial relationships or prioritize key rows (e.g., totals, year summaries).
- When given too many noisy sub-query results, it can still fail to promote the correct chunk.

There's still no strong notion of "row importance" or hierarchy in current reranking logic.

LLM Struggles with Financial Reasoning

While LLMs like GPT-4 excel at language tasks, their numeric reasoning can be brittle:

- Minor changes in phrasing can lead to incorrect aggregation logic.
- Multi-hop reasoning over table + narrative is non-trivial, even with structured prompts.

Year Filtering Helps, but Isn't Robust

Regex-based year detection improved precision, but:

- It sometimes filters out valid multi-year data if the match is too strict.
- Doesn't handle relative references well (e.g., "last year", "compared to 2020").

Engineering Bottlenecks

- Embedding step (FAISS + row chunking) is computationally heavy and not incremental, rebuilding the index takes time.
- Latency (~9s/query) is tolerable for evaluation but not ideal for production settings.
- No caching or persistent memory, so all LLM calls are stateless, which affects follow-up questions or conversational flow.

Evaluation Edge Cases

The evaluation logic handles numeric formatting (e.g., commas, %, $), but:

- It doesn't support multi-answer questions (e.g., "List revenue and net income for 2020").
- Evaluation still falls back on fuzzy matching, which may underestimate correct answers if formatting varies heavily.
  The pipeline solves a lot, but is still bounded by retrieval quality and the limits of prompt-based LLM reasoning in financial contexts. Fixing the recall/precision trade-off and incorporating structured table understanding (e.g., via table parsing models) are promising next steps.

7. Future Work

While the current system demonstrates promising accuracy and architectural robustness, several enhancements can significantly improve performance, robustness, and real-world applicability.

Smarter and Incremental Indexing

- Dynamic Updates: Current FAISS index is static. In future, enable incremental additions (e.g., new financial filings).
- Column-Aware Chunking: Instead of splitting only by rows, chunking could incorporate column semantics, so that queries on specific metrics (e.g., "EBITDA") map more directly to relevant entries.

Hybrid Retrieval (Dense + Sparse)

Dense vector search captures semantic meaning but lacks specificity. Future plans:

- Integrate BM25 or keyword filtering alongside FAISS to narrow down chunks based on exact matches (e.g., metric names, years).
- Use hybrid scores to combine semantic and lexical relevance, improving precision.

Enhanced Sub-Query Generation

Sub-queries currently rely on a prompt-fed LLM call. Improvements could include:

- Fine-tuning a lightweight T5 model to handle question decomposition faster and more consistently.
- Adding deduplication or ranking logic to filter noisy sub-queries automatically.

Financial Table Understanding

Large language models still struggle with complex table reasoning. To mitigate this:

- Use table-specific encoders (e.g., TAPAS, FinQANet) to encode financial tables more structurally.
- Annotate tables with schema labels (headers, totals, deltas) to give LLMs more context.

Retrieval-Aware Reranking

The reranker today works post-retrieval. Instead:

- Integrate a learned retriever-reranker model, where reranking can feed back into improving retrieval.
- Explore contrastive learning techniques to fine-tune better chunk representations with relevance supervision.

Conversational Memory and Context

Currently, each query is stateless. In real-world applications like investor chat or research assistants:

- Add memory tracking or conversation state (e.g., LangChain memory modules).
- Carry over table references or filters between turns (e.g., "Now show the same for 2021").

Advanced Evaluation

Improve the evaluator to handle:

- Multi-part answers, lists, or aggregated values (e.g., “total of A + B”).
- Confidence scoring, so low-quality outputs can be flagged or rerouted.

Could also track:

- Top-k accuracy: is the answer somewhere in top-3 completions?
- Relevance attribution: how well the retrieved document supports the answer?

Deployment and Optimization

- Latency Reduction: Optimize prompt templates, use async inference, or switch to faster local LLMs (e.g., Mistral, Ollama).
- Caching: Save sub-query generations and retrievals for frequently asked questions.
- Cloud-Hosted Index: Move FAISS to a hosted vector DB (e.g., Pinecone, Weaviate) for production scalability.

Future work focuses on precision-first retrieval, better financial comprehension, and conversational intelligence. With additional table-aware models and evaluation improvements, this system can evolve into a robust domain-specific financial assistant.

8. Conclusion

This project began with a simple goal: build a retrieval-augmented generation (RAG) pipeline capable of answering complex financial questions from the ConvFinQA dataset. Along the way, it became clear that traditional RAG wasn't enough, especially when precision, structure, and domain-specific reasoning were critical.

To address this, I focused heavily on engineering enhancements that improved each stage of the pipeline:

- Breaking financial tables into individual rows to increase retrieval granularity.
- Generating sub-queries to expand semantic coverage.
- Applying regex-based year filtering for precision targeting.
- Separating context into structured narrative and tabular components for clarity.
- Using reranking (via Cohere) to prioritize the most relevant chunks.
- Designing LLM prompts that clearly convey structured input to reduce hallucinations.

Through rigorous evaluation, I demonstrated that retrieval quality, not generation, is the primary bottleneck in financial QA systems. When ground-truth documents were provided (oracle setup), accuracy jumped to nearly 80%, reinforcing this insight.

This project taught me how important architecture, evaluation, and careful prompt engineering are when deploying LLMs in real-world, high-stakes domains like finance. It also reaffirmed the importance of granular chunking, retrieval precision, and structured reasoning when working with structured data.

There's still more to explore: hybrid retrievers, schema-aware encoders, better reranking, and long-term conversational memory. But I'm proud of the system I've built and confident that it lays a strong foundation for domain-specific, trustworthy QA in financial contexts.