ConvFinQA Question Answering System

1. Method, Design & Architecture

This project is a retrieval-augmented generation (RAG) pipeline I built to answer complex financial questions using the ConvFinQA dataset. It's broken into a few modular components:

Core Components:

- **Data Parsing:** converted the raw ConvFinQA JSON files into a structured CSV. Each entry combines the table and its narrative context into a unified text block. Tables were converted into **markdown format** because it's easier for LLMs to parse compared to raw formats like CSV or HTML. Research and practical usage have shown that markdown tables significantly improve LLM comprehension, especially for reasoning over structured data.
- **Indexing:** I chunked each document at the row level (one row = one chunk) and embedded them using SentenceTransformer (E5/BGE). All the embeddings are stored in a FAISS index to enable efficient vector-based retrieval.
- **RAG Pipeline:** When a question comes in:
    - I first generate the sub-queries from the original question using llm.
    - Use those sub-queries to drive dense retrieval from the FAISS index.
    - Results are then passed through Cohere's reranker to reorder them by relevance.
    - The final context is split into two parts: tabular data and narrative descriptions.
    - A structured prompt feeds both context types to the LLM, which generates the answer.

Evaluation Framework:

I use the provided gold-standard QA pairs from ConvFinQA for evaluation. Metrics include:

- Answer accuracy (LLM output vs. gold),
- Precision and recall at both retrieval and reranking stages,
- End-to-end latency.

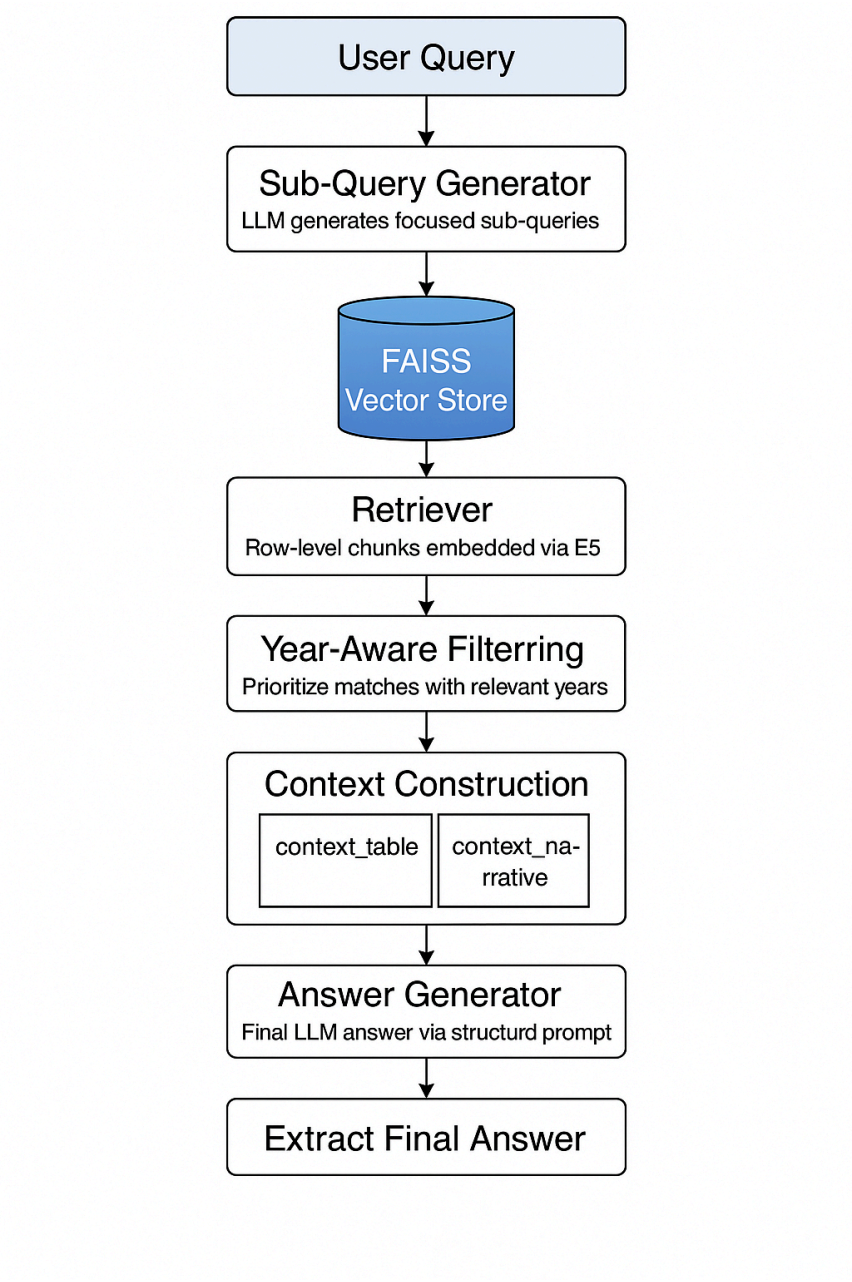

Figure 1: End-to-End Architecture of the ConvFinQA Question Answering System

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.

2. Why Basic RAG Wasn't Enough

Initial experiments using vanilla RAG (retrieving entire table + context blobs) didn't work out. Problems I ran into:

- **Loss of Detail:** Financial questions are often super specific like asking for numbers from a single year or row. Large document chunks blurred those fine details.

- **Mixed Inputs:** LLMs struggled when table and narrative content were blended. They need structure to reason properly.

- **Low Retrieval Precision:** Dense retrievers returned results that were semantically close, but not numerically relevant. In early runs, precision was basically zero.

3. Fixes & Improvements

To fix those issues, I implemented a few key changes:

- **Row-Level Chunking:** Tables are split into individual rows during indexing, which made retrieval far more targeted for year/metric-specific questions.

- **Sub-query Generation:** The system breaks the user's question into multiple focused search phrases. This step massively improved the signal-to-noise ratio during retrieval.

- **Year-Aware Filtering:** I added logic to detect years in the question using regex and bias retrieval toward entries that match.

- **Dual Context Structure:** Retrieved content is organized into:

    - context_table: the raw financial data
    - context_narrative: descriptive explanation
      Both are passed separately to the LLM for better reasoning.

- **Cohere Reranking:** Used rerank-english-v3.0 to improve result ordering. This made a noticeable difference in downstream answer quality.

- **Math Matching for Eval:** Added light numeric comparison logic to tolerate formatting differences in answers (e.g., "$9,000" vs. "9000").

- **Prompt Refinement:** I designed prompts that clearly separate the question, table, and narrative. This gives the LLM everything it needs to reason more accurately.

4. Evaluation & Metrics

I evaluated the full QA pipeline using the official ConvFinQA training set. The evaluation script looks at both the quality of the final answer and the retrieval/reranking steps along the way.

Answer Accuracy:

This checks if the predicted answer matches the gold answer. I use:

- Exact match when possible.
- Fuzzy numeric matching to handle minor formatting differences.

Average Accuracy: 41.42% (over 500 examples, no ground-truth retrieval)

Retrieval + Reranking Metrics:

To break down retrieval performance further, I looked into:

- **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

As seen, reranking improves precision significantly, even if recall slightly drops. This helps LLMs focus on fewer but more relevant contexts. This directly improves answer quality.

5. Oracle Evaluation:

use_ground_truth_retrieval = True
When the correct document is always given (oracle setup), accuracy jumps to 79.58%, showing that retrieval quality is the current bottleneck, not generation.

6. Limitations:

The reranker is still imperfect, even with good sub-queries, it sometimes misses.
Regex-based year filtering may exclude valid multi-year documents.
Math reasoning is weak if answers require multi-row aggregation.

8. Future Enhancements:

The current RAG system works decently, but there's still a lot of room to improve accuracy, robustness, and reasoning. Here's what I'd like to work on next:

First, the table parsing could be a lot cleaner. Using a more structured parser like something Pandas-based or even a layout-aware model like LayoutLM would help. I'd also normalize things like currency, percentages, and large numbers right at ingestion time, stuff like turning "$9,000" into just 9000—so the model has less to guess.

Next, multi-row reasoning is still a weak spot. I'd like to add a lightweight math layer that can handle basic operations : sum, difference, ratios on the retrieved rows before we even pass anything to the LLM. This could be done with a symbolic math tool like SymPy or even a small custom logic layer to pre-process that kind of reasoning.

The reranker could definitely be smarter. Right now, it's generic. I want to try training or fine-tuning a reranker specifically for financial Q, maybe even build a simple relevance classifier that understands financial table structures and how they relate to different types of questions.

Sub-query generation could also get a boost. Right now it's just a prompt. I want to try a few-shot examples or fine-tune a small model on financial QA data to generate tighter sub-queries. Also thinking about adding retrieval feedback so if no good results come back, the system can try adjusting the query automatically on the fly.

Hybrid retrieval is another big one. Right now it's only dense (FAISS), but adding something like BM25 could help catch useful lexical matches that dense methods miss. Then I'd combine and re-rank the two result sets for better coverage overall.

I'd also like to build in some kind of confidence estimation. Just a lightweight score that helps tell when the LLM is guessing vs. when it's confident. That could help with flagging low trust answers or even deciding when to expand the prompt window dynamically.