Text-Only vs. Table-Aware Retrieval for Financial Question Answering on Public SEC Filings

Gefei Zhao gzhao46@wisc.edu
Chengyun Jia cjia36@wisc.edu
Chenyu Wei cwei66@wisc.edu
Yuliang Wu wu738@wisc.edu
STAT 453, Spring 2026
Team 03

Abstract

Financial question answering over SEC 10-K filings is a strong stress test for retrieval-augmented generation because many answers depend on grounded numerical evidence rather than fluent paraphrase alone. This report documents the final `sec-financial-rag/` snapshot of our project, which builds a three-stage pipeline around a FinanceBench-aligned corpus. Step 1 processes 64 retained 10-K filings and exposes open-box corpus metadata; Step 2 implements BM25, dense, hybrid, and table-aware retrieval; Step 3 evaluates 112 retained FinanceBench questions with a fixed generator, `openai/gpt-oss-20b`, question-specific metadata filters, adaptive evidence bundles, and full manual review. Although the title reflects the original text-only framing, the final BM25, dense, and hybrid baselines retrieve from a combined text-and-table chunk pool. In the final committed benchmark, dense retrieval is the strongest overall system, achieving Recall@10 of 0.973, MRR of 0.809, answer_accuracy of 0.621, and evidence_support_quality of 0.585. Across all four retrievers, the dominant reviewed failure mode is not random hallucination but unnecessary abstention: the model often outputs "insufficient evidence" even when the retrieved evidence already contains enough information to answer. The final results therefore support two conclusions at once: retrieval quality remains a bottleneck in financial QA, and structure-aware retrieval is promising, but in this committed implementation strong single-stream dense retrieval over the combined chunk pool outperforms the explicitly table-aware retriever end to end.

1 Introduction

Retrieval-augmented generation (RAG) systems are often evaluated on flattened text corpora, but financial filings contain a large amount of structured evidence in balance sheets, statements of cash flows, segment tables, and note disclosures. In SEC Form 10-K filings, a question may look natural-language on the surface while still depending on a specific fiscal-period column, a row inside a financial statement, or a calculation that combines values across multiple retrieved blocks. That makes financial QA a useful setting for asking whether standard retrieval is enough or whether retrieval needs to preserve structure more explicitly.

The original project question was straightforward: *does table-aware retrieval improve financially grounded question answering over public SEC filings when the answer generator is held fixed?* We kept that question constant, but the committed implementation evolved relative to the earliest draft. The final report is based on the curated FinanceBench 10-K processed subset preserved in `sec-financial-rag/`, not the earlier all-company benchmark draft in our working archive. It is also important to be precise about what is being compared in the final code. The original proposal emphasized text-only baselines versus table-aware retrieval, but the committed Step 3 runner gives

BM25, dense, and hybrid access to a combined text-and-table chunk pool, while `table_aware` preserves separate text and table retrieval streams before fusion. The final benchmark is therefore best understood as a comparison between single-stream baselines and an explicitly structure-aware fusion retriever.

This final snapshot makes four concrete contributions. First, it preserves a reproducible, FinanceBench-aligned 10-K corpus with open-box metadata that explains exactly which documents were processed and which were skipped. Second, it implements four comparable retrievers over the same downstream QA task: BM25, dense, hybrid, and table-aware. Third, it evaluates the four systems under one fixed generator, `openai/gpt-oss-20b`, with question-conditioned retrieval filters and a full manual review layer. Fourth, it preserves enough reviewer-facing artifacts in GitHub form to audit the benchmark without shipping the full raw archive.

The rest of the report follows the structure encouraged by the project guideline. Section 2 reviews related work. Section 3 describes the dataset and the three-stage pipeline. Section 4 summarizes the final experiment setup. Section 5 explains how the evaluation and labels work. Section 6 presents the results, with metric interpretation first. Section 7 discusses failure modes and compares our findings to prior work. Sections 8–10 cover limitations, team contributions, and conclusions. The appendix at the end of this report provides reproducibility notes and AI-assistance disclosure.

2 Related Work

Recent work on financial QA and financial RAG consistently argues that retrieval quality is a central bottleneck because financial evidence is long-range, numerically dense, and often structured. FinanceBench introduced an open-book benchmark for financial question answering over public-company filings and showed that answer quality depends heavily on whether the system finds the correct supporting evidence rather than only generating fluent prose [1]. That benchmark is directly relevant here because the final `sec-financial-rag/` snapshot uses a filtered FinanceBench 10-K subset rather than a custom benchmark.

More recent benchmark and systems work pushes the same argument in a more structure-sensitive direction. FinRAGBench-V emphasizes visually grounded and multimodal financial retrieval, showing that page layout and structured evidence matter in addition to plain text [2]. FinSage treats financial RAG as a system-design problem in which retrieval, evidence selection, and answer grounding are separable engineering choices rather than one monolithic model behavior [3]. TableRAG similarly argues that heterogeneous documents require retrieval schemes that preserve structural relationships instead of flattening everything into a single text stream [4].

At the same time, the literature also justifies strong non-specialized baselines. Hybrid retrieval remains a common practical choice because lexical search is strong on exact terms, abbreviations, and finance-specific strings, while dense retrieval is better at broader semantic matching. That prior work set our expectation going into the project: structure-aware retrieval should help especially on numeric or table-heavy questions, but the size of the gain depends on benchmark design, chunking policy, evidence formatting, and how answer generation reacts to retrieved evidence. Our final findings match some parts of that expectation and differ from others, which becomes important in the discussion.

Corpus or Benchmark Quantity	Value
FinanceBench questions in source file	150
Referenced FinanceBench documents	84
Processed 10-K filings retained by Step 1	64
Questions retained for Step 3 evaluation	112
Skipped documents	20
Skipped questions	38
Flagged filings in `quality_report.json`	2
Question types in final benchmark	43 mixed, 52 numeric, 12 table, 5 text
Manual-review rows in override CSV	448

Table 1: Final benchmark summary from the committed `github_final/` artifacts.

3 Dataset and Pipeline

3.1 Final Benchmark and Corpus

The final committed benchmark is built from the FinanceBench open-source question file and document metadata preserved under `sec-financial-rag/01_data_collection/data/financebench/`. The Step 1 pipeline then processes the corresponding 10-K source documents and records what was retained in the open-box corpus handoff under `sec-financial-rag/01_data_collection/data/processed/open_box/`. In the current snapshot, all retained processed documents are `10K` filings and the source document kind recorded in the manifest is `financebench_pdf`. The GitHub snapshot intentionally omits the bulky raw PDF corpus and per-filing processed trees, but it preserves the metadata necessary to understand what the benchmark contains.

The most important final counts come from `financebench\_coverage.json`, `latest\_corpus\_manifest.json`, and the Step 3 summary tables. FinanceBench contributes 150 total questions in the local source file. After filtering to documents that are both 10-Ks and successfully processed by Step 1, the final benchmark keeps 112 questions over 64 filings. Because no model is trained or tuned, this project uses the retained FinanceBench 10-K subset as an evaluation benchmark rather than a train/test split. The open-box coverage report records 20 skipped documents and 38 skipped questions. Question types in the final benchmark are highly imbalanced: 43 mixed questions, 52 numeric questions, 12 table questions, and only 5 text questions. This imbalance matters for interpretation later, because the benchmark is driven much more by mixed and numeric behavior than by pure text or pure table questions.

Step 1 also records parser quality flags. In the committed snapshot, two filings remain flagged rather than removed: `AMCOR_2023_10K`, which is missing core Items 1 and 1A in its recovered section structure, and `PFIZER_2021_10K`, which is missing Items 1, 1A, 7, and 8. Keeping them is useful because they expose how robust the later retrieval and QA stages are to imperfect preprocessing rather than assuming clean inputs throughout.

3.2 Three-Stage Pipeline

The final repository is organized as a three-stage pipeline.

Step 1: FinanceBench-aligned corpus construction. The code under `sec-financial-rag/01_data_collection/src/sec_rag/` ingests the FinanceBench document set, parses filings, and writes compact open-box artifacts such as `dataset\_index.csv`, `dataset\_index.json`, `quality\_report.json`, and `latest\_corpus\_manifest.json`. The GitHub snapshot is intentionally lightweight: it keeps corpus-level handoff files but omits the bulky raw and per-filing trees

needed for a full local rebuild.

Step 2: Retrieval. The code under `sec-financial-rag/02_retrieval/` builds comparable retrieval systems over the processed corpus. The shared presets file and README specify the final chunking defaults used by the public notebooks: text chunks allow up to 450 tokens with 100-token overlap, while table chunks are serialized in groups of 8 rows with up to 500 nearby-context characters and `only_data_tables=True`. Dense and hybrid retrieval use `BAAI/bge-base-en-v1.5`, and reciprocal-rank fusion uses `retrieve_depth=50` and `rrf_k=60`.

Step 3: QA evaluation. The code under `sec-financial-rag/03_qa_evaluation/` builds retrievers, formats evidence, runs the generator, scores retrieval, and writes reviewer-facing outputs. The final Step 3 runner is more question-conditioned than the earlier draft benchmark: ticker, year, and accession filters are enabled from question metadata, text questions use `top_k_context=3`, non-text questions expand to at least 5 evidence blocks, and non-text questions use a deeper retrieval pool with a minimum of 20 retrieved candidates. The fixed generator for the committed snapshot is `openai/gpt-oss-20b`.

4 Method and Experimental Setup

4.1 Retriever Design in the Final Snapshot

The committed Step 3 comparison includes four retrievers:

- **BM25**: lexical retrieval over the combined chunk pool.
- **Dense**: embedding-based retrieval over the combined chunk pool.
- **Hybrid**: reciprocal-rank fusion over BM25 and dense retrieval on the combined chunk pool.
- **Table-aware**: separate text and table retrieval streams fused after retrieval.

That design detail is important. In the final code, BM25, dense, and hybrid are not pure narrative-text baselines; they retrieve over the union of text chunks and table chunks. The `table_aware` system therefore differs mainly by preserving separate text and table streams and by using structure-aware evidence selection, not by being the only retriever allowed to see tables. This later implementation choice likely narrows the gap between dense or hybrid retrieval and the explicitly structure-aware retriever.

4.2 Final Run Configuration

The final run configuration combines Step 2 chunking defaults with Step 3 question-conditioned retrieval and prompting. Table 2 summarizes the settings that matter most for interpretation.

The prompt is also stricter than in the earlier working version. It tells the model to output `insufficient evidence` only when the provided evidence does not support a reliable answer, and it explicitly instructs the model to answer if the needed figures or statements are already present in the evidence blocks. That change matters because many reviewed failures in the final benchmark are unnecessary abstentions rather than outright hallucinations.

5 Evaluation Protocol

The final benchmark combines automatic retrieval-side evaluation with a full manual review layer.

Parameter	Value Used	Source of Truth
Text chunking	450 max tokens, 100 overlap	Step 2 presets
Table chunking	8 rows per group	Step 2 presets
Table context cap	500 chars	Step 2 presets
Table filtering	`only_data_tables=True`	Step 2 presets
Dense model	`BAAI/bge-base-en-v1.5`	Step 2 and Step 3
Dense batch size	32	Step 2 and Step 3
Fusion depth / RRF constant	50 / 60	Step 2 and Step 3
Base retrieval depth	`retrieval_top_k=10`	Step 3 run metadata
Non-text minimum retrieval pool	20	Step 3 run metadata / code
Text question evidence bundle	3 blocks	Step 3 run metadata / code
Non-text minimum evidence bundle	5 blocks	Step 3 run metadata / code
Question-side metadata filters	ticker, year, accession enabled	Step 3 run metadata / code
Generator model	`openai/gpt-oss-20b`	Step 3 run metadata
Generation style	`max_new_tokens=256`, `temperature=0.1`, `do_sample=False`	Step 3 config

Table 2: Key settings used by the final committed `github_final/` experiment.

Retrieval is evaluated question by question using document-constrained matching in `qa\_eval.py`. A retrieved row counts as a hit only after it first matches the expected `financebench_doc_name`. After that, hits are awarded through one of three routes: exact or nearby page match, normalized overlap with gold evidence text, or numeric-answer anchoring for quantitative questions.

Question types are not taken directly from raw FinanceBench categories. Instead, `question\_loader.py` derives four proposal-style categories—`text`, `table`, `numeric`, and `mixed`—from the question text, answer form, and evidence structure. The committed `question\_type\_overrides.csv` file is currently empty, so the final reported question-type breakdowns are the direct output of those heuristics.

Manual review uses the reviewer-facing packs under `sec-financial-rag/03_qa_evaluation/results\_financebench\_10k/`. For the final committed snapshot, `manual\_review\_overrides\_financebench\_10k.csv` contains 448 reviewed rows, which corresponds to all 112 questions across all four retrievers. Reviewers inspect the question text, gold answer, gold evidence summary, generated answer, citations, and retrieved evidence snippets.

6 Results

6.1 How to Read the Reported Metrics

Before comparing retrievers, it is important to define what the reported metrics mean.

- **Recall@k** asks whether useful supporting evidence appears anywhere in the top k retrieved results.
- **MRR** (mean reciprocal rank) measures how early the first useful hit appears. Higher MRR means the first relevant block tends to be ranked closer to the top.
- **relaxed_string_match** and **numeric_tolerance_match** are helper signals only. They are useful for debugging, but they are not the final quality judgment.

Retriever	Recall@10	MRR	Citation Hit Rate	Answer Accuracy	Evidence Support	Mean Retrieval (s)	Mean Total (s)
BM25	0.955	0.745	0.312	0.522	0.482	0.132	5.585
Dense	0.973	0.809	0.455	0.621	0.585	0.508	5.714
Hybrid	0.973	0.796	0.455	0.576	0.549	0.582	5.806
Table-aware	0.955	0.644	0.304	0.585	0.527	0.590	5.309

Table 3: Overall performance summary from `combined_summary_by_retriever.csv` in the committed Step 3 results.

- **citation presence**, **citation validity**, and **citation hit rate** indicate whether answers cite evidence and whether those cited blocks correspond to retrieval hits.
- **manual_correctness** uses three labels: `correct`, `partial`, and `wrong`.
- **evidence_support** uses three labels: `good`, `partial`, and `wrong`.
- **answer_accuracy** is the mean of the manual-correctness scores with `correct`=1, `partial`=0.5, and `wrong`=0.
- **evidence_support_quality** is the mean of the evidence-support scores with `good`=1, `partial`=0.5, and `wrong`=0.

Two interpretation rules are especially important. First, `answer_accuracy` is a reviewed response-quality score, not a pure "true answer recovery" rate. A system can receive credit for a defensible abstention when the shown evidence is genuinely too weak, and it can also lose credit when it refuses to answer even though the evidence is already sufficient. Second, `answer_accuracy` should not be interpreted alone when discussing user usefulness. It needs to be read together with `evidence_support_quality`, citation behavior, and retrieval metrics. A system that answers often but grounds weakly is different from a system that answers slightly less often but grounds much more cleanly.

6.2 Overall Comparison

Table 3 summarizes the most important metrics from `combined\_summary\_by\_retriever.csv`. Dense is the strongest overall system in the final committed snapshot. It achieves the best MRR, ties for the best Recall@10, and has the best overall `answer_accuracy` and `evidence_support_quality`. Hybrid remains close on several metrics, especially Recall@10 and citation hit rate. Table-aware retrieval is no longer the top overall system. BM25 remains the fastest retriever in pure retrieval latency, but because generation dominates end-to-end latency, the lowest mean total runtime in the saved snapshot actually belongs to table-aware.

The main empirical picture is therefore more mixed than the original project title might suggest. The structure-aware retriever is competitive, but in this final benchmark the dense single-stream baseline is best overall. That difference from the early working draft is one of the most important results of the final report.

6.3 Breakdown by Question Type

Question type matters a great deal. Table 4 shows that mixed and numeric questions dominate the benchmark and remain the most informative subgroups. Dense is strongest on the large mixed subset and on the small table subset. Table-aware has the best answer_accuracy and evidence-support quality on the numeric subset, but it underperforms dense on mixed questions and has a

Retriever	Question Type	Questions	Recall@10	Answer Accuracy	Evidence Support
BM25	mixed	43	0.930	0.453	0.407
BM25	numeric	52	0.981	0.538	0.529
BM25	table	12	0.917	0.667	0.542
BM25	text	5	1.000	0.600	0.500
Dense	mixed	43	1.000	0.616	0.593
Dense	numeric	52	0.981	0.538	0.519
Dense	table	12	0.833	0.917	0.750
Dense	text	5	1.000	0.800	0.800
Hybrid	mixed	43	0.953	0.570	0.535
Hybrid	numeric	52	1.000	0.500	0.500
Hybrid	table	12	0.917	0.833	0.708
Hybrid	text	5	1.000	0.800	0.800
Table-aware	mixed	43	0.977	0.547	0.477
Table-aware	numeric	52	0.981	0.558	0.548
Table-aware	table	12	0.750	0.750	0.583
Table-aware	text	5	1.000	0.800	0.600

Table 4: Question-type breakdown from `combined_summary_by_question_type.csv`. The `text` and `table` subsets are very small and should be interpreted cautiously.

Retriever	Wrong or Partial Rows	`insufficient_evidence_but_answer_available`
BM25	54	53
Dense	43	42
Hybrid	48	47
Table-aware	47	46

Table 5: Failure-note summary from `manual_review_overrides_financebench_10k.csv`. The dominant note for every retriever is unnecessary abstention on answerable cases.

substantially lower MRR overall. The `text` and `table` subgroups are tiny (5 and 12 questions), so those rows should be read as descriptive rather than definitive.

6.4 Failure Analysis by Retriever

The manual-review overrides expose one shared pattern very clearly. Across all four retrievers, the dominant review note is `insufficient_evidence_but_answer_available`. In other words, the most common reviewed failure is not unsupported fabrication; it is unnecessary abstention on answerable questions. Those failures appear most often on numeric and mixed questions that require the model to extract a value from the shown evidence, combine a small number of values, or trust a retrieved table row enough to answer. Table 5 summarizes this shared pattern.

BM25. BM25's common failure pattern is lexical near-miss retrieval. It often returns evidence that is topically related but not decisive enough to answer the question. These errors cluster in numeric and mixed questions where the answer depends on a specific cash-flow, balance-sheet, or segment-growth row. In the BM25 review pack, failed cases such as 3M's FY2018 capital expenditure question and Activision Blizzard's fixed-asset-turnover question retrieve nearby MD&A discussion or general financial-statement prose instead of the exact row that contains the needed figures. The shared characteristic is that the evidence looks plausibly relevant, but the decisive line is missing.

Dense. Dense retrieval has fewer failures overall, but its dominant mistakes still follow the

same unnecessary-abstention pattern. The review packs show several cases in which the retrieved evidence already contains the needed figures or very close supporting statement text, yet the model still outputs `insufficient evidence`. These failures are common in answerable numeric and mixed questions that require a short computation or synthesis across a few blocks. The likely bottleneck is not total retrieval collapse but answer-use behavior: the system finds semantically relevant evidence and then fails to convert it into a final answer.

Hybrid. Hybrid remains a strong baseline, but its failures suggest that lexical-plus-dense fusion does not automatically solve the abstention problem. The shared pattern is similar to dense retrieval—numeric and mixed questions remain the hardest—but some hybrid bundles include redundant or less decisive blocks, which can leave the final evidence package noisier than the best dense run. That is consistent with the final summary tables: hybrid is competitive on Recall@10 and citation hit rate, but its end-to-end gains over dense are limited.

Table-aware. Table-aware retrieval fails for a different reason than BM25 even when the final answer is still missing. It often retrieves structurally relevant table chunks, but those chunks are sometimes hard for the generator to use because the serialization is awkward, the row group is incomplete, or the headers are generic. In the review pack, some failures show clearly table-shaped evidence that is still not legible enough to support a confident answer, such as 3M's FY2018 capital expenditure question and several balance-sheet or multi-step mixed questions. The shared characteristic is not the absence of structural evidence, but structural evidence that remains difficult to read and synthesize after chunking and serialization.

Taken together, the four failure profiles point to one high-level synthesis. The common failure across all systems is answerable evidence not being turned into a final answer. The main differences are where the bottleneck appears: BM25 most often misses the decisive exact evidence, dense often finds usable evidence but does not always use it, hybrid remains strong but sometimes produces noisier fusion bundles, and table-aware preserves more structure but still needs cleaner serialization and stronger answer-use behavior.

7 Discussion and Comparison to Related Work

Our findings are similar to the prior literature in several ways. First, they support the central lesson of FinanceBench and related financial QA work: retrieval quality is a major bottleneck, and grounded evidence matters more than fluent answer wording alone [1]. Second, they match the general intuition in FinSage and TableRAG that structured or heterogeneous evidence changes the retrieval problem in meaningful ways [3, 4]. Third, like much of the financial RAG literature, we find that numeric and mixed questions are harder than simpler text questions and that answer quality depends heavily on whether the retrieved evidence can actually be used by the generator [2].

At the same time, our final results differ from a simple "table-aware always wins" story. In the final committed snapshot, dense retrieval is the best overall system, and table-aware retrieval is not the top end-to-end method. There are several plausible reasons for that difference. The first is benchmark design: the final benchmark is a filtered FinanceBench 10-K processed subset rather than a benchmark built entirely to expose table-centric failures. The second is that the final Step 3 runner uses ticker, year, and accession filters from question metadata. That narrows the retrieval problem and reduces the relative advantage of broader structure-aware search. The third is that the final baselines are not pure text-only systems. BM25, dense, and hybrid retrieve from the combined chunk pool that already includes table chunks, so the explicitly table-aware method is competing against stronger baselines than the project title alone might suggest.

The current evidence-selection and prompting strategy is another likely cause. The final prompt is stricter about when abstention is allowed and more willing to answer when the needed figures are already present. In addition, non-text questions receive deeper retrieval pools and larger evidence bundles. Those changes seem to help strong dense retrieval substantially. Table-aware retrieval still has a meaningful structural idea behind it, but the final manual-review packs show that structure alone is not enough if table chunks remain awkwardly serialized, truncated, or paired with less useful context. In other words, preserving structure helps only if that structure remains usable by the downstream answer generator.

Our findings also fit part of the hybrid-retrieval literature while differing in detail. It is still true that hybrid systems are strong practical baselines. In our benchmark, hybrid is close to dense overall and competitive on several retrieval metrics. However, dense slightly outperforms hybrid on the final overall combination of retrieval rank, answer_accuracy, and evidence_support_quality. That suggests fusion is not automatically better under every filtered benchmark, evidence-selection policy, or prompt regime.

The most balanced interpretation is therefore not that prior structure-aware work is wrong. Rather, our final snapshot shows that benchmark design, metadata filtering, chunk construction, table serialization, and answer-use behavior all influence whether structure-aware retrieval produces end-to-end gains. Structured retrieval remains a meaningful design direction, but the committed implementation here still needs better evidence formatting and better answer synthesis before its structural advantage consistently converts into the best final QA performance.

8 Limitations

The final report has several important limitations.

- The manual-review layer uses a single durable override file rather than multi-reviewer adjudication.
- The public `sec-financial-rag/` snapshot omits the bulky raw PDFs, per-filing processed trees, and intermediate QA exports, so not every low-level artifact is directly visible in GitHub.
- The final benchmark is highly imbalanced by question type, especially with only 12 table questions and 5 text questions.
- The final Step 3 benchmark is question-conditioned rather than unrestricted open-corpus retrieval because ticker, year, and accession filters are enabled from question metadata.
- The table-aware retriever depends on serialized table chunks, which still lose some usability even when they preserve more structure than flattened narrative text.

9 Team Contributions

The final contribution summary follows the proposal's role split.

- **Gefei Zhao**: coordinated SEC/FinanceBench data handling, preprocessing, and open-box corpus organization; contributed to benchmarking and final interpretation.
- **Chengyun Jia**: implemented and validated BM25-related retrieval components and helped compare baseline retrieval behavior.

- **Chenyu Wei**: implemented dense and hybrid retrieval components and contributed to evaluation and comparison logic.
- **Yuliang Wu**: focused on table-aware retrieval, error analysis, result interpretation, and coordination of the final report.

All team members contributed to the literature review, experiment design, result interpretation, and final writing.

10 Conclusion

This report documents the final `sec-financial-rag/` version of our project rather than the earlier draft benchmark. The committed pipeline processes a FinanceBench-aligned subset of 64 10-K filings, evaluates 112 retained questions, and compares four retrievers under one fixed generator and a full manual-review layer.

The main empirical result is clear: in the final committed snapshot, dense retrieval is the strongest overall configuration. It ties for the best Recall@10, achieves the best MRR, and leads the benchmark on both `answer_accuracy` and `evidence_support_quality`. Hybrid remains a strong competitor, BM25 remains the fastest retriever in pure retrieval latency, and table-aware retrieval remains meaningful but is not the top-performing end-to-end system in this snapshot.

The most important qualitative lesson is that retrieval quality still drives financial QA, but many failures are now failures of evidence use rather than total retrieval collapse. Across all four systems, the dominant reviewed error is unnecessary abstention on answerable questions. That makes the final benchmark more informative than a simple winner-take-all ranking: it shows that strong retrieval is necessary, structure-awareness is promising, and cleaner evidence presentation may be the next key step for turning retrieved financial evidence into better final answers.
Project repository: https://github.com/JasonJia12/sec-financial-rag

References

[1] Pranab Islam, Anand Kannappan, Douwe Kiela, Rebecca Qian, Nino Scherrer, and Bertie Vidgen. FinanceBench: A New Benchmark for Financial Question Answering. arXiv preprint arXiv:2311.11944, 2023.

[2] Suifeng Zhao, Zhuoran Jin, Sujian Li, and Jun Gao. FinRAGBench-V: A Benchmark for Multimodal RAG with Visual Citation in the Financial Domain. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 4215–4249, 2025.

[3] Xinyu Wang et al. FinSage: A Multi-aspect RAG System for Financial Filings Question Answering. arXiv preprint arXiv:2504.14493, 2025.

[4] Xiaohan Yu, Pu Jian, and Chong Chen. TableRAG: A Retrieval Augmented Generation Framework for Heterogeneous Document Reasoning. arXiv preprint arXiv:2506.10380, 2025.

[5] U.S. Securities and Exchange Commission. Form 10-K. https://www.sec.gov/files/form10-k.pdf.

Appendix

A AI-Assistance Disclosure

We used Codex interactively throughout the project as a coding, planning, and writing collaborator. In particular, we used GPT-5.4 inside Codex to help develop implementation plans for each stage of the project, then used those plans to guide implementation work for Step 1 data collection, Step 2 retrieval, and Step 3 QA evaluation.

Codex helped us inspect files, summarize existing artifacts, explain pipeline behavior, organize documentation, and draft report-supporting text. The important project choices remained human-directed: the project team chose the research question, benchmark framing, three-stage pipeline structure, retrievers to compare, evaluation metrics, manual-review labels, result interpretation, limitations, and final report wording.

A.1 Human-Directed Decisions

The project team made the important research and reporting decisions while using Codex as support. The most important human-directed choices were specific to the three project stages rather than generic writing preferences.

Step 1 Data Collection

Humans chose to build the corpus around FinanceBench-linked SEC 10-K filings instead of creating a new question set from scratch. The team decided to rely on local FinanceBench PDFs as the stable input source, filter the benchmark to filings that were actually processed by Step 1, and preserve skipped-document and skipped-question accounting instead of hiding coverage losses. The team also chose to keep parser quality flags visible in the final artifact, including filings with missing core items, and to make the open-box handoff files such as `dataset_index.csv`, `financebench_coverage.json`, `latest_corpus_manifest.json`, and `quality_report.json` the public evidence for what the corpus contains.

Step 2 Retrieval

Humans chose the four retrieval methods to compare: BM25, dense, hybrid, and table-aware. The team decided that Step 2 should stay retrieval-only, so answer generation, citation analysis, and manual QA scoring would be handled later in Step 3. The team also chose to build both text chunks and table chunks, use `BAAI/bge-base-en-v1.5` for dense retrieval, and treat table-aware retrieval as separate text/table branch fusion rather than as an extra generation method. This is why the final report distinguishes the explicitly table-aware retriever from the BM25, dense, and hybrid baselines that retrieve over a combined text-and-table chunk pool.

Step 3 QA Evaluation

Humans chose to evaluate only the retained processed FinanceBench questions, use question metadata filters for ticker, year, and accession when available, and hold answer generation fixed with `openai/gpt-oss-20b`. The team decided to use adaptive evidence bundles for non-text questions, review generated answers manually with answer-correctness and evidence-support labels, and preserve manual-review labels in a durable override CSV. Humans also interpreted the failure-note pattern rather than treating automatic metrics as the whole result: the final conclusion is that

dense retrieval wins overall in the committed benchmark, table-aware retrieval remains meaningful but not best overall, and unnecessary abstention is the dominant reviewed failure mode.

The key instructions preserved in our AI-assistance workflow were concrete review behaviors. Humans reviewed generated implementation plans before using them, inspected code and output artifacts rather than accepting summaries blindly, selected which result files belonged in the final GitHub snapshot, verified report metrics against committed CSV summaries, and approved the final framing of the report, appendix, limitations, and AI-assistance disclosure.

We use the repository's `AGENTS.md` file in the current project to record the stage boundaries, reproducibility expectations, and human-oversight principles that guide Codex's behavior.

The three implementation-plan prompts developed with Codex are included below for transparency.

B Step 1 Data Collection Build Plan Request

B.1 Summary

Build Step 1 of the SEC financial RAG project from scratch as a reproducible data-collection and preprocessing stage. Use `V2_full_artifact/` and `sec-financial-rag/` only as reference materials to understand the desired final behavior and artifact format.

The goal is to produce a FinanceBench-aligned 10-K corpus that can support later retrieval and QA experiments.

B.2 Key Requirements

- Start by inspecting the reference folders and existing project documentation to infer the expected Step 1 contract.
- Use FinanceBench metadata as the document-selection source.
- Keep the default corpus focused on annual filings: `10K` and `10K_ANNUAL`.
- Support local FinanceBench PDFs as the reliable source input.
- Prefer SEC/EDGAR HTML only when available and usable.
- Produce standardized outputs for each filing:
  - cleaned text
  - extracted 10-K sections
  - extracted table metadata
  - table CSV files
  - filing-level metadata
- Produce corpus-level open-box artifacts:
  - dataset index
  - quality report
  - corpus manifest
  - FinanceBench coverage summary
  - skipped-document report

B.3 Implementation Tasks

- Create a clean Python package for Step 1 with a CLI entry point.
- Implement FinanceBench JSONL loading for questions and document metadata.
- Join questions to documents by `doc_name`.
- Filter unsupported document types deterministically and record why they were skipped.
- Stage raw source files into a consistent `data/raw/filings/` layout.
- Parse PDF and/or HTML filings into a consistent processed format.
- Extract 10-K item sections such as Items 1, 1A, 7, and 8 when possible.
- Extract tables into metadata plus CSV files for downstream table-aware retrieval.
- Generate a `dataset_index.csv` that Step 2 can load without needing to understand raw FinanceBench files.
- Add quality flags for weak parsing cases, especially missing core 10-K items.
- Write README and data-location documentation so the Step 1 workflow is easy to rerun and inspect.

B.4 Test Plan

- Unit test FinanceBench loading and validation.
- Unit test document filtering and skipped-document reasons.
- Unit test PDF/HTML parsing on small fixtures.
- Unit test section extraction and table extraction.
- Integration test one small end-to-end run that creates raw, processed, and open-box artifacts.
- Verify that the generated dataset index can be consumed by the retrieval stage.
- Run a final smoke test comparing generated corpus-level counts against the reference artifacts.

B.5 Assumptions

- The implementation should prioritize reproducibility and reportability over maximizing corpus size.
- The final Step 1 contract should be designed for Step 2 retrieval and Step 3 QA evaluation, not just standalone scraping.

C Step 2 Retrieval Build Plan Request

C.1 Summary

Build Step 2 of the SEC financial RAG project from scratch as a retrieval-only stage over the Step 1 corpus. Use `V2_full_artifact/` and `sec-financial-rag/` only as reference materials to understand the desired final behavior and artifact format.

The goal is to compare four retrieval methods over the FinanceBench-aligned 10-K corpus: `bm25`, `dense`, `hybrid`, and `table_aware`. Do not include answer generation, citation grading, or manual QA scoring in Step 2.

C.2 Key Requirements

- Start by inspecting the Step 1 handoff files and existing project documentation to infer the expected Step 2 contract.
- Load the Step 1 corpus through `dataset_index.csv`, then resolve each filing's `sections.json`, `tables.json`, and linked table CSV files.
- Preserve FinanceBench provenance fields when available, including document name, document type, source kind, accession, ticker, year, and question IDs.
- Build standardized retrieval chunks for:
  - text sections
  - row-group table chunks
  - combined text-and-table retrieval pools
  - separate text and table streams for table-aware retrieval
- Implement exactly four reported retrieval methods:
  - BM25 lexical retrieval
  - dense embedding retrieval
  - hybrid BM25-plus-dense retrieval
  - table-aware retrieval with separate text/table branches
- Keep Step 2 retrieval-only and report retrieval-side metrics only.
- Centralize shared defaults so notebooks and scripts use the same chunking, model, and fusion settings.

C.3 Implementation Tasks

- Create a clean Python package for Step 2 with reusable modules for loading, chunking, retrieval, evaluation, and shared presets.
- Implement a corpus loader that reads Step 1 open-box artifacts and resolves all paths relative to the Step 1 dataset root.
- Support both legacy Step 1 schema fields and FinanceBench-enriched schema fields.

- Build text chunks from filing sections, with long-section windowing and boilerplate filtering.
- Build table chunks from table metadata and CSV files, preserving headers, row groups, nearby context, section metadata, and page metadata when available.
- Implement BM25 retrieval over any chunk list with ticker and metadata filtering.
- Implement dense retrieval using sentence-transformer embeddings and a FAISS flat index.
- Implement hybrid retrieval using reciprocal rank fusion over BM25 and dense rankings.
- Implement table-aware retrieval by retrieving text chunks and table chunks separately, then fusing the branch rankings.
- Add retrieval evaluation utilities for result tables, query-level metrics, and method comparison summaries.
- Create full-dataset notebooks for BM25, dense, hybrid, and table-aware retrieval using consistent project paths and shared presets.
- Write README documentation explaining inputs, retrieval methods, defaults, metrics, and notebook usage.

C.4 Retrieval Defaults

- Use `top_k=10` for standard retrieval examples.
- Use text chunks of about `450` tokens with `100` token overlap.
- Use table row-group chunks with `8` rows per group.
- Use a table context cap of about `500` characters.
- Use `BAAI/bge-base-en-v1.5` as the dense embedding model.
- Use reciprocal rank fusion with `retrieve_depth=50` and `rrf_k=60`.
- Keep table-aware retrieval as a fusion of separate text and table branches, not as a fifth reported method.

C.5 Test Plan

- Unit test the loader on Step 1-style fixture data, including FinanceBench-enriched metadata and older schema fallback.
- Unit test text chunking, including boilerplate skipping and long-section windowing.
- Unit test table chunking, including row grouping, context preservation, failed-table skipping, and data-table filtering.
- Unit test BM25 retrieval with metadata and ticker filters.
- Unit test dense retrieval with mocked or small deterministic embeddings where possible.
- Unit test hybrid and table-aware rank fusion behavior.

- Unit test evaluation outputs for stable result columns, Recall@k, MRR, hit rate, and latency fields.
- Integration test a small end-to-end retrieval run from a fixture Step 1 dataset through all four retrievers.
- Smoke test the notebooks or notebook-equivalent scripts against the generated Step 1 corpus.

C.6 Assumptions

- Step 2 consumes Step 1 outputs and should not scrape, download, or preprocess filings itself.
- Step 2 should compare retrieval methods only; final answer generation and answer-quality evaluation belong to Step 3.
- The final Step 2 contract should be designed for Step 3 QA evaluation, not just standalone notebook demos.

D Step 3 QA Evaluation Build Plan Request

D.1 Summary

Build Step 3 of the SEC financial RAG project from scratch as an end-to-end question answering and evaluation stage on top of the Step 2 retrievers. Use `V2_full_artifact/` and `sec-financial-rag/` only as reference materials to understand the desired final behavior and artifact format.

The goal is to run the FinanceBench-aligned 10-K question set across the four Step 2 retrievers: `bm25`, `dense`, `hybrid`, and `table_aware`, then generate answers, evaluate retrieval and answer quality, and produce reviewer-facing outputs.

D.2 Key Requirements

- Start by inspecting the Step 1 and Step 2 handoff files and existing project documentation to infer the expected Step 3 contract.
- Load FinanceBench questions from the Step 1 metadata files and keep only questions whose documents were processed in Step 1.
- Preserve question metadata, including FinanceBench ID, document name, document type, company, ticker, year, accession number, gold answer, and gold evidence.
- Derive proposal-style question types: `text`, `table`, `numeric`, and `mixed`.
- Support optional question-type overrides from a small CSV file.
- Build or reuse the four Step 2 retrievers without adding extra reported retriever methods.
- Use question-specific retrieval filters from metadata, including ticker, year, and accession when available.
- Format retrieved evidence into inspectable evidence blocks for generation.
- Run a fixed answer generator with prompts that require evidence-only answers and citations.

- Produce retrieval results, answer results, evaluation tables, summary tables, and manual-review artifacts.
- Keep manual-review labels durable across reruns through a small override CSV.

D.3 Implementation Tasks

- Create a clean Python package for Step 3 with reusable modules for question loading, prompting, generation, QA running, evaluation, reporting, and manual review.
- Implement a question loader that joins FinanceBench question rows, document metadata, and Step 1 `dataset_index.csv`.
- Filter the question set to processed 10-K documents only.
- Implement question-type classification heuristics using question text, answer form, reasoning metadata, and evidence structure.
- Implement a runner configuration object that controls dataset paths, output paths, retriever settings, generation settings, retrieval depth, evidence bundle size, and metadata filters.
- Build corpus assets from Step 1 and Step 2 chunks, with optional cache support for repeated notebook runs.
- Build a retriever registry exposing exactly `bm25`, `dense`, `hybrid`, and `table_aware`.
- For each question, build a retrieval request, retrieve candidates, select generation evidence, format evidence blocks, and call the generator.
- Use larger evidence bundles and deeper retrieval pools for non-text questions when needed.
- Implement prompt formatting with explicit instructions to use only retrieved evidence, cite evidence blocks, and answer `insufficient evidence` only when support is missing.
- Implement a lazy Hugging Face generation wrapper with `openai/gpt-oss-20b` as the default generator.
- Parse generated outputs into answer text, citations text, cited evidence numbers, and raw model output.
- Implement evaluation metrics for retrieval, answer matching, numeric matching, citations, manual correctness, evidence support, and runtime.
- Implement reporting helpers that save per-retriever outputs and combined summaries.
- Implement manual-review helpers that generate compact review queues, markdown review packs, and durable override application.
- Create notebooks for the all-retriever run, per-retriever reruns, and final comparison summaries.
- Write README and manual-review documentation explaining inputs, workflow, outputs, labels, and notebook usage.

D.4 QA Defaults

- Use the four Step 2 retrievers: `bm25`, `dense`, `hybrid`, and `table_aware`.
- Use `top_k_context=3` for text questions.
- Use at least `5` evidence blocks for non-text questions when available.
- Use `retrieval_top_k=10` as the baseline retrieval depth.
- Use a deeper candidate pool for non-text questions, with a minimum of about `20` retrieved candidates.
- Use ticker, year, and accession metadata filters when available.
- Use `BAAI/bge-base-en-v1.5` for dense retrieval inherited from Step 2.
- Use `openai/gpt-oss-20b` as the default generator.
- Use low-temperature deterministic generation settings for reproducible answers.

D.5 Output Artifacts

- Save per-retriever `question_manifest.csv/json`.
- Save per-retriever `retrieval_results.csv/json`.
- Save per-retriever `answer_results.csv/json`.
- Save per-retriever `evaluation.csv/json`.
- Save per-retriever `summary_by_retriever.csv/json`.
- Save per-retriever `summary_by_question_type.csv/json`.
- Save per-retriever `run_metadata.json`.
- Save per-retriever `manual_review_queue.csv/json`.
- Save per-retriever `manual_review_pack.md`.
- Save a combined `manual_review_index.md`.
- Save combined summary tables across all four retrievers.

D.6 Test Plan

- Unit test FinanceBench question loading, processed-document filtering, and metadata joins.
- Unit test question-type classification and question-type overrides.
- Unit test retrieval request construction with ticker, year, and accession filters.
- Unit test retriever registry construction and confirm exactly four methods are exposed.
- Unit test evidence selection for text, numeric, table, and mixed questions.

- Unit test prompt and evidence-block formatting.
- Unit test generated-answer parsing, including answer text and cited evidence numbers.
- Unit test retrieval evaluation using page matches, nearby-page matches, evidence overlap, and numeric answer anchors.
- Unit test answer evaluation for relaxed string matching, numeric tolerance matching, citation validity, and citation hit rate.
- Unit test manual-review queue creation, manual-review pack creation, and override application.
- Integration test one small end-to-end QA run with a stub retriever and stub generator.
- Smoke test notebook-equivalent scripts for all four retrievers and the final comparison summaries.

D.7 Assumptions

- Step 3 consumes Step 1 and Step 2 outputs and should not scrape filings or redefine retrieval methods.
- Step 3 owns answer generation, QA evaluation, manual review, and reporting.
- Manual correctness labels use `correct`, `partial`, and `wrong`.
- Evidence support labels use `good`, `partial`, and `wrong` or compatible wrong-style labels already present in the review data.
- The final Step 3 contract should support both automated metrics and human review.

E Reproducibility Guide

This appendix documents the current `sec-financial-rag/` snapshot that supports the final report. The main report focuses on interpretation, while this appendix records where the relevant artifacts live and what a reader would need to inspect or rerun the benchmark.

E.1 Repository Layout

The final report is based on the root-level project folder together with the curated GitHub snapshot:

```
RAG Project/
  final_project_report.tex
  sec-financial-rag/
    01_data_collection/
    02_retrieval/
    03_qa_evaluation/
```

Within `sec-financial-rag/`, the most useful entry points are:

- `sec-financial-rag/README.md`: top-level project overview.

- `sec-financial-rag/01_data_collection/README.md`: Step 1 corpus construction and open-box handoff.
- `sec-financial-rag/02_retrieval/README.md`: Step 2 retrieval methods and notebook layout.
- `sec-financial-rag/03_qa_evaluation/README.md`: Step 3 QA workflow and committed outputs.
- `sec-financial-rag/03_qa_evaluation/MANUAL_REVIEW_GUIDE.md`: final review workflow and label semantics.

E.2 Key Artifact Map

Path	Role
`sec-financial-rag/01_data_collection/data/processed/open_box/financebench_coverage.json`	Coverage summary: total questions, processed questions, skipped documents, skipped questions.
`sec-financial-rag/01_data_collection/data/processed/open_box/latest_corpus_manifest.json`	Step 1 retained filing list and corpus run summary.
`sec-financial-rag/01_data_collection/data/processed/open_box/quality_report.json`	Parser quality notes and flagged filings.
`sec-financial-rag/02_retrieval/src/step2_retrieval/presets.py`	Step 2 chunking and retrieval defaults.
`sec-financial-rag/03_qa_evaluation/src/step3_qa/qa_runner.py`	Final Step 3 runner behavior, including metadata filters and dynamic evidence selection.
`sec-financial-rag/03_qa_evaluation/src/step3_qa/question_loader.py`	Heuristic derivation of the proposal-style question types.
`sec-financial-rag/03_qa_evaluation/src/step3_qa/qa_eval.py`	Metric definitions and score mappings for manual labels.
`sec-financial-rag/03_qa_evaluation/data/manual_review_overrides_financebench_10k.csv`	Durable manual-review labels across all four retrievers.
`sec-financial-rag/03_qa_evaluation/results_financebench_10k/combined_summary_by_retriever.csv`	Main overall comparison table.
`sec-financial-rag/03_qa_evaluation/results_financebench_10k/combined_summary_by_question_type.csv`	Question-type breakdown used in the report.

Table 6: Key files used to write and verify the final report.

E.3 What Was Intentionally Omitted

The public `sec-financial-rag/` snapshot is intentionally lightweight. It omits:

- the full FinanceBench PDF corpus,
- raw staged filings and per-filing processed trees,

- tests, caches, macOS clutter, and bulky rerun outputs,
- intermediate Step 3 exports such as `answer_results.*`, `retrieval_results.*`, `evaluation.*`, and editable review queues.

Those omissions do not prevent interpretation of the final benchmark, but they do mean a full rerun requires restoring the omitted raw inputs.

E.4 Current Configuration Summary

Setting	Committed Final Value
Processed filings / evaluated questions	64 / 112
Question types	43 mixed, 52 numeric, 12 table, 5 text
Text chunking	450 max tokens, 100 overlap
Table chunking	8 rows per group, 500 context chars, `only_data_tables=True`
Dense model	`BAAI/bge-base-en-v1.5`
Fusion parameters	`retrieve_depth=50`, `rrf_k=60`
Base retrieval depth	`retrieval_top_k=10`
Non-text minimum retrieval pool	20
Text evidence bundle size	3 blocks
Non-text minimum evidence bundle size	5 blocks
Metadata filters	ticker, year, accession enabled
Generator	`openai/gpt-oss-20b`

Table 7: Condensed configuration summary for the final committed experiment.

E.5 Recommended Verification Workflow

To verify the final report against the committed snapshot:

1. Read `sec-financial-rag/README.md` for the high-level structure.
2. Read the three stage READMEs in order: Step 1, Step 2, then Step 3.
3. Check `financebench_coverage.json`, `latest_corpus_manifest.json`, and `quality_report.json` to confirm dataset counts and flagged filings.
4. Check `combined_summary_by_retriever.csv` and `combined_summary_by_question_type.csv` to confirm the result tables.
5. Check `manual_review_overrides_financebench_10k.csv` and `MANUAL_REVIEW_GUIDE.md` to confirm the label semantics and failure-note patterns.