Evaluation of dsRAG on FinanceBench

1. Summary

This report presents a comprehensive evaluation of **dsRAG** pipeline configurations on the **FinanceBench** open-book QA dataset. The evaluation assesses how different document chunking methods, embedding models, re-rankers, and inference models impact accuracy, latency, and cost. Key findings include:

- **Accuracy vs. Model Size:** Pipelines using large language models (LLMs) like Anthropic Claude 4.0 and OpenAI GPT-4 achieved the highest accuracy (≈76–81% on FinanceBench), about **30 percentage points higher** than pipelines using smaller language models (SLMs) (≈43–45%). However, LLM-based pipelines incur significantly higher computational cost and latency (2–3× slower inference).
- **Top-Performing Pipeline:** The best-performing configuration used **Claude 4.0** for both document chunking and final inference, with a strong re-ranker and large embeddings. It answered **122 out of 150 questions correctly (81.3% accuracy)**, demonstrating superior handling of complex financial queries. This came at the highest cost and runtime (~27 minutes for the full test set).
- **Cost-Effective Pipeline:** For data-sensitive or cost-constrained deployments, a fully local pipeline using **DeepSeek-R1-Distill Qwen-7B** (7B parameter model) for retrieval, **MiniLM-L12-v2** embeddings, and a lightweight **Voyage 2.5B re-ranker** achieved ~45% accuracy. While substantially lower in accuracy, this pipeline had no external API calls, faster total inference (~12–14 minutes), and minimal token costs, making it suitable for internal use cases.
- **Retrieval & Re-ranking:** Incorporating advanced retrieval techniques and re-rankers significantly improved answer accuracy. The dsRAG pipeline's **Relevant Segment Extraction (RSE)** improved recall by grouping related chunks, mitigating incomplete evidence issues. A learned re-ranker (Voyage 2.5 lite) boosted accuracy by ~21 points over no re-ranking (to ~57% vs ~36%).
- **Recommendations:** A **hybrid deployment strategy** is advised – use cost-efficient SLM pipelines (augmented with strong re-rankers and retrieval tweaks) for high-volume or sensitive queries, and reserve LLM pipelines (Claude, GPT-4) for complex, high-stakes financial analyses. Additionally, **fine-tuning** domain-specific models and **knowledge distillation** from LLMs are recommended to narrow the performance gap while controlling costs.
- Overall, dsRAG demonstrated robust performance on FinanceBench, especially when leveraging top-tier LLMs. The following sections detail the performance by component category, comparative insights, best pipeline configurations, and recommendations for future improvements.

2. Category-wise Analysis

In this section, we examine the performance and behavior of each major pipeline component category — from the type of language model used for inference (small vs. large) to the choice of

embedding model and re-ranker. The goal is to understand how each category contributes to overall system accuracy, latency, and cost.

Small Language Models (SLMs)

SLMs refer to smaller-parameter models (such as 7B–13B) that can be deployed locally. In our evaluation, the primary SLM used was **DeepSeek-R1-Distill Qwen-7B**, a 7-billion parameter model distilled for the finance domain. Pipelines with this SLM had **substantially lower accuracy**, stabilizing around **43–45%** on FinanceBench. The SLM struggled with complex multi-step reasoning and numeric calculations due to its limited capacity and smaller context window. For example, SLM pipelines failed to answer ~58.6% of the benchmark's numeric questions correctly, compared to only ~20.7% failure for LLM pipelines. They also often required more aggressive retrieval augmentation (e.g. using RSE to combine chunks) to compensate for their weaker reasoning ability. On the positive side, SLM-based pipelines offered **faster inference** and **lower operational cost**. The average total runtime for 150 queries was ~730–850 seconds (≈12–14 minutes) with an SLM, roughly 2× faster than LLMs. There are also **no API costs** for local SLMs, an important consideration for **data-sensitive or cost-sensitive environments** where keeping data on-premise is required. In summary, SLMs provide cost-efficiency and data privacy, but at the expense of significant accuracy loss and difficulty with complex reasoning.

Large Language Models (LLMs)

LLMs in this evaluation include state-of-the-art models like **Anthropic Claude 4.0 Sonnet** and **GPT-4** (denoted as GPT-4o in our results). These models delivered the **highest accuracy**, about **76–81.33%** on the FinanceBench QA set. LLM pipelines excelled at understanding long and complex financial documents and performing multi-step numeric reasoning. They handled **long evidence chains** with less reliance on heavy pre-processing or chunk enrichment and were more robust on mixed numeric/text questions. Notably, Claude 4.0 slightly outperformed GPT-4 on this dataset (Claude achieved ~76% vs GPT-4's ~75% accuracy) – it was the single best model tested. However, these gains came with **higher costs and latency**. LLM runs were **2–3× slower** than SLMs on average, with a full 150-query run taking ~1600 seconds (~26–27 minutes). The token-based API pricing for processing lengthy financial reports and generating answers is also non-trivial – for instance, using GPT-4 for chunking and inference incurred sizable costs (in our knowledge base creation, processing ~368 reports with GPT-4 cost on the order of \$137, and inference calls further add to this). LLM usage may also encounter operational limits like API rate limits or context length constraints. In short, **LLMs delivered superior accuracy and reasoning** ability – making them ideal for **high-stakes, precision-critical analyses** – but require careful consideration of **cost, latency, and throughput limitations** in deployment.

Embedding Models

The embedding model transforms text chunks into vector representations for retrieval. We evaluated both **proprietary** embedding models and **open-source** ones: - **OpenAI text-embedding-3-large** (a high-dimensional transformer embedding): This model provided the **most accurate retrieval**, which translated into higher end-to-end accuracy. Pipelines using the large embedding had on average ~3–4% higher accuracy than those using a smaller embedding model (e.g. 66.9% vs 63.6%) in our tests. However, this performance comes at a cost: generating

embeddings for the entire corpus cost roughly **\$9** in OpenAI usage, and it increased the upfront chunking time (slower throughput during knowledge base construction due to the larger model). - **OpenAI text-embedding-3-small:** a smaller embedding model (lower dimensional). This was **faster and cheaper** (only **\$1.95** total embedding cost for the corpus). The trade-off was slightly weaker retrieval – pipelines with the small embedding saw a few percentage points drop in accuracy. This model may be sufficient for simpler information needs but showed lower recall for complex queries. - **all-MiniLM-L12-v2 (MiniLM-L12):** an open-source, locally hosted embedding model. This model offered a strong balance of performance and cost-efficiency. While its absolute accuracy was lower in our raw results (~39% average, since it was used mainly with weaker SLM pipelines), qualitatively MiniLM demonstrated solid semantic recall. More importantly, being local, it **incurs no token cost** and can embed text quickly on a GPU. For domain-specific deployment, fine-tuned or higher-dimensional open embeddings like MiniLM provide a viable **cost-free alternative** to proprietary APIs. - **Other embeddings:** We also tried a **Voyage-large-2** embedding model (an internal high-capacity model) and an **Infogrid Stella v2** model. The Voyage-large embedding performed similarly to OpenAI's small embed (yielding ~62% pipeline accuracy on average). The Stella-base-v2 model (open-source) underperformed relative to MiniLM, suggesting it may not be as well-suited to financial text.

Overall, investing in a stronger embedding model improved retrieval recall and thus final answer accuracy. For example, using the **text-embedding-3-large** model in retrieval was a key factor in all of the top 10 highest-accuracy pipelines. Meanwhile, a well-chosen local embedding (like MiniLM-L12) can drastically cut costs while still maintaining reasonable performance.

Re-ranking Models

Re-rankers reorder the initially retrieved documents to promote truly relevant passages to the top. In our evaluation, **re-ranking proved crucial** for boosting accuracy. Without re-ranking, the base retrieval often returned some irrelevant or less-useful chunks, leading to only ~33–36% accuracy. Adding any learned re-ranker yielded a significant jump in accuracy (typically +10–20 percentage points). We evaluated a few re-ranker models: - **Voyage rerank-2.5B:** a 2.5-billion parameter re-ranker (with a "lite" distilled variant). The **Voyage-2.5 lite** model gave the best overall results, providing the **highest accuracy gains vs. latency**. Pipelines using Voyage-2.5 lite achieved the top accuracies (~57% average, versus 36% with none. Notably, the lightweight variant slightly **outperformed the full 2.5B model** in our tests (possibly due to being distilled on domain data), while also being faster. This re-ranker is optimized for speed, making it an excellent choice to improve quality without much added runtime. - **Cohere Re-rank (english-v3.0):** a hosted neural re-ranker from Cohere. It was also effective – pipelines with Cohere re-rankers saw large accuracy improvements (e.g. +9 points on one SLM pipeline, from 36% to ~45%). However, the Cohere model was observed to be **slightly slower** than Voyage for similar gains, and as an external API, it incurs its own cost. In latency-sensitive scenarios, this might be a consideration. - **BAAI-bge reranker-base:** an open-source re-ranker (based on BGE embeddings). This model provided **moderate improvements** – lifting accuracy to ~51.7% on average – but generally did not match the top-tier re-rankers. For example, when paired with a strong LLM, the BGE re-ranker pipeline reached ~68% accuracy, versus ~75% with Voyage under the same conditions. Its advantage is that it can be self-hosted (no API costs). BGE could be a useful lightweight option if computational resources for re-ranking are very limited. - **No Re-ranker:** for comparison, the baseline retrieval-alone pipelines had the lowest accuracy (~36%). Many answers were missed because the correct

evidence chunk was ranked lower and thus excluded from the LLM's context. This underscores that a competent re-ranking stage is essential in an open-book QA pipeline like dsRAG.

In summary, **re-ranking is a high-leverage component** for FinanceBench QA. The Voyage 2.5-lite model is recommended for its strong accuracy boost and efficiency. Even for cost-sensitive deployments, an open re-ranker (Cohere's free tier or a local model) should be employed, as it substantially improves the quality of retrieved context and final answers.

Inference Models

This category refers to the final answer-generation model (the LLM or SLM that reads the retrieved documents and answers the question). Performance varied widely across inference models: - **Anthropic Claude 4.0 Sonnet:** This model achieved the **highest accuracy of all**. In our evaluation Claude answered ~114/150 questions correctly (~76.3%) when paired with strong retrieval. Claude showed strength in complex reasoning and lengthy responses, making it ideal for detailed financial Q&A. The trade-off is Claude's API cost, which was the highest among models, and its slower generation time (leading to the longest run time). - **OpenAI GPT-4 (GPT-4o):** We denote the GPT-4 API as GPT-4o. It delivered nearly as strong performance as Claude, about **75% accuracy**, and excelled particularly in numeric computations and precise answers. GPT-4's cost per token is high, but it may require slightly fewer tokens than Claude for the same answer (due to more concise style), partially mitigating cost differences. Both Claude and GPT-4 are best suited for **high-stakes analyses** where accuracy is paramount. - **GPT-4o-mini:** This refers to a moderately-sized model we used for inference; it is roughly equivalent to an **open-source 13B–30B class model or a distilled GPT-3.5-level model**. Interestingly, with an optimized pipeline (LLM-assisted chunking, etc.), this model achieved up to **74% accuracy** on FinanceBench – rivaling the larger proprietary models. On average across all its runs, it scored ~52% (since many runs with simpler retrieval dragged the average down). This suggests that **a mid-tier model can approach top performance if supported by excellent retrieval and possibly fine-tuning**. GPT-4o-mini also ran faster and cheaper than GPT-4 (e.g. if it corresponds to GPT-3.5, the token cost was an order of magnitude lower than GPT-4). This model represents a middle-ground for organizations seeking a balance between cost and accuracy. - **DeepSeek-33B Chat:** This is a **33B-parameter in-house model** fine-tuned for chat/inference. As a fully local LLM, it achieved about **58% accuracy** on FinanceBench – substantially higher than the 7B model, but still ~15–18 points below GPT-4. Its runtime was about 20 minutes for 150 queries, faster than GPT-4 but slower than the 13B model. The 33B model could answer most factual questions correctly, but struggled with the hardest multi-hop reasoning that the 65B+ class models handled. Still, this result is promising: with domain fine-tuning, a ~30B scale model can potentially exceed 60% accuracy, offering a **viable on-premises solution** for moderate accuracy needs. - **DeepSeek-R1-Distill Qwen-7B:** The 7B distilled model had the lowest accuracy at ~32–33% when used as the final inference engine (with no larger model involvement). It frequently failed on questions requiring arithmetic or synthesis of multiple facts. We also tested a **Mistral 7B** instruct model, which performed similarly (~24–28% accuracy). These small models simply lack the capacity for complex financial QA without further training. They should be considered only in scenarios where **privacy and cost are absolutely critical** and one is willing to accept a high error rate. In such cases, aggressive retrieval (to feed the model exact answer snippets) and fine-tuning on QA format can help squeeze out better performance.

To summarize, **larger inference models yield dramatically better results**, but mid-sized models can be viable with the right support. Figure 3 (below) highlights the accuracy differences across inference model classes. The challenge is to close the gap between the 7B and 70B scale models via techniques like fine-tuning and knowledge distillation (discussed later), so that more organizations can deploy capable QA systems without relying on proprietary APIs.

3. Comparative Insights

This section synthesizes the above results to highlight key comparative insights across different pipeline configurations:

SLMs vs. LLMs – Accuracy and Trade-offs: There is a clear **accuracy gap** between small local models and large state-of-the-art models: LLM pipelines answered about 30% more questions correctly than SLM pipelines (an average of ~75% vs ~45% accuracy). This gap was especially pronounced on complex, multi-step problems – LLMs solved many numeric and reasoning-intensive queries that SLMs could not. However, this accuracy comes with **3–4× higher cost** (in token usage) and ~2× longer latency. SLM pipelines, by contrast, are **faster and far cheaper** to operate, making them attractive for high-volume or privacy-sensitive use cases despite their lower accuracy. In practice, organizations may choose a hybrid approach: use SLMs for straightforward or internal queries, and call on LLMs for the hardest questions. This balances cost and performance.

Impact of Embedding Model on Retrieval: The choice of embedding model had a **significant effect on retrieval quality**, which trickled down to final accuracy. Using a powerful embedding (like OpenAI's text-embedding-3-large) improved the chances that the relevant document segments were retrieved and ranked highly. We observed pipelines with the large embedding outperform those with a smaller embedding by ~3-5 percentage points accuracy, all else equal. That said, the smaller embedding was much cheaper and faster, illustrating a *precision vs. efficiency* trade-off. Notably, using a good **local embedding model (MiniLM-L12)** yielded surprisingly competitive results – while we didn't pair MiniLM with the largest LLMs in this eval, its strong semantic performance suggests it could replace the expensive API embeddings with only a minor hit to accuracy. For domain-specific deployments, an open embedding model fine-tuned on domain text might achieve nearly the same retrieval performance as a general-purpose large embed, at virtually no cost.

Value of Re-ranking (RAG Pipeline Ablation): Adding a learning-to-rank stage provided a **clear boost** to the system. In a head-to-head comparison, the best re-ranker (Voyage 2.5B lite) raised final accuracy to ~57.5% on average, versus ~36% with no re-ranking. This demonstrates that initial vector retrieval alone may miss or mis-order relevant info, especially when questions span multiple pieces of evidence. A re-ranker addresses this by using a cross-encoder to carefully score each candidate passage in the query context. Interestingly, even a relatively small re-ranker model (the "lite" 2.5B) was enough to confer most of the benefits – we did not see a huge difference between it and a larger 2.5B model variant, implying that re-ranker quality saturates with moderate model size for this task. It's also worth mentioning that the **Cohere re-ranker** improved accuracy similarly to Voyage in many cases, reinforcing that the concept (not just one specific model) of re-ranking is valuable. The key insight is that **retrieval should not stop at a raw**

vector search – applying a learned re-ranker greatly improves precision, which directly translates to better answer correctness.

4. Best Pipelines

We now highlight the top-performing pipeline configurations identified in our evaluation, along with a contrast to the most cost-efficient configuration:

Highest-Accuracy Pipeline

The pipeline with the **highest accuracy** combined **Anthropic Claude 4.0** for both chunking and inference, **OpenAI's text-embedding-3-large** for embedding, and the **Voyage rerank-2.5 lite** model for re-ranking. This configuration answered **122 out of 150 questions correctly (81.3% accuracy)**, the peak performance observed. The use of Claude 4.0 (an extremely capable 100B+ parameter model) allowed the system to handle complex reasoning and lengthy context with ease, while the large embedding and effective re-ranker ensured the model was fed highly relevant, comprehensive context for each query. However, this pipeline was also the **most expensive**: it required calling a premium LLM API for every document chunk (during KB construction) and every query, incurring substantial token costs (hundreds of dollars in a full deployment scenario). It was also the slowest, with an average per-query latency of ~10.8 seconds (total ~27 minutes for 150 Qs). **Table 1** (below) summarizes this and other top pipelines. Notably, the second-best pipeline used GPT-4 (labeled GPT-4o) in place of Claude and achieved 75.0% accuracy at slightly lower cost and latency, indicating GPT-4 is a comparably strong choice. The third-ranking pipeline used a smaller open model ("GPT-4o-mini") for inference and still reached 74% accuracy, demonstrating that with the right retrieval setup, even a mid-sized model can perform impressively well.

Cost-Efficient Pipeline

For scenarios where **minimal cost and data privacy** are paramount, we identified a pipeline that, while lower accuracy, uses only local or inexpensive models. This configuration employed **DeepSeek-R1-Distill Qwen-7B** (7B) as the core LLM, along with **all-MiniLM-L12-v2** embeddings and the **Voyage 2.5 lite re-ranker**. In practice, we ran this pipeline with an assist from a 13B model (GPT-4o-mini) for final answer generation to reach about **45% accuracy** – but if needed, one could use the 7B model itself for inference (yielding ~33% accuracy) to avoid any external dependency. The strength of this pipeline is that **all components can run on-premise**: the 7B model and MiniLM embed are open-source, and the Voyage lite re-ranker can be self-hosted. There are **no API costs**, and sensitive financial data never leaves the local environment. The runtime was also quite reasonable (~12–13 minutes for all queries, or ~5 seconds per query on average). The obvious downside is accuracy – at 45%, this pipeline answers fewer than half of the questions correctly, struggling especially with those requiring deep reasoning or precise numeric computation. We recommend this lightweight pipeline only for low-priority queries or initial proof-of-concept deployments. With targeted **fine-tuning on financial QA data**, it is possible this pipeline's accuracy could be improved into the 50–60% range while preserving its cost advantages.

Table 1. Top 10 dsRAG pipeline configurations by accuracy, along with their key components and total inference time on 150 queries. All top-performing pipelines used the large embedding model (text-embedding-3-large). Claude 4.0 and GPT-4 (GPT-4o) delivered the highest accuracy, while the GPT-4o-mini model offered a good trade-off between accuracy and speed. Voyage re-rankers (especially the lite variant) were common in all top pipelines, underlining their importance.

Rank	Chunking Model	Re-ranker	Inference Model	Accuracy	Total Inference Time
1	Claude 4.0	Voyage2.5-lite	Claude 4.0	81.3% (122/150)	1618 s (~10.8 s/Q)
2	GPT-4o (GPT-4 API)	Voyage2.5-lite	GPT-4o	75.0% (112/150)	940 s (~6.3 s/Q)
3	GPT-4o	Voyage2.5-lite	GPT-4o-mini (13B)	74.0% (111/150)	1051 s (~7.0 s/Q)
3	GPT-4o-mini (13B)	Voyage2.5-lite	GPT-4o-mini (13B)	74.0% (111/150)	1093 s (~7.3 s/Q)
3	GPT-4o	Voyage2.5 (full)	GPT-4o	74.0% (111/150)	950 s (~6.3 s/Q)
6	Claude 4.0	Voyage2.5 (full)	Claude 4.0	73.3% (110/150)	1652 s (~11.0 s/Q)
6	GPT-4o	Voyage2.5 (full)	GPT-4o-mini (13B)	73.3% (110/150)	1053 s (~7.0 s/Q)
8	Claude 4.0	Cohere v3.0	Claude 4.0	72.0% (108/150)	1659 s (~11.1 s/Q)
8	GPT-4o-mini (13B)	Voyage2.5 (full)	GPT-4o-mini (13B)	72.0% (108/150)	1033 s (~6.9 s/Q)
10	GPT-4o	Cohere v3.0	GPT-4o	70.0% (105/150)	921 s (~6.1 s/Q)

Table 2. Model-wise breakdown of average accuracy, latency, and cost considerations by stage. Each entry shows the performance when that component was used, averaged across relevant pipelines: - *Embedding Models:* Larger, high-dimensional embeddings yield better retrieval (e.g. text-embedding-3-large ~66.9% avg accuracy) at the cost of API usage and slower processing. Local embeddings (MiniLM) are cost-free but slightly lower in recall.
- *Re-rankers:* Using a re-ranker substantially improves accuracy (+15–21 pts). Voyage-2.5-lite offered the best accuracy (~57.5%) with minimal latency overhead. Cohere's re-ranker was effective (~45.3%) but adds external API latency. BGE was a decent open alternative (~51.7%). Not re-ranking leaves accuracy very low (~36%).
- *Inference Models:* Claude 4.0 was the top performer (~71.5% avg), slightly above GPT-4 (~69.8%). A mid-sized GPT-4o-mini model reached ~52% on average (up to 74% max), showing promise for cheaper deployment. A local 33B model hit ~58%, whereas the 7B model trailed at

~28%. Inference latency ranges from ~0.5–1.0 sec/Q for 7B models up to ~10+ sec/Q for GPT-4/Claude.

Stage	Model	Avg Accuracy	Remarks (Cost & Latency)
Embedding	text-embedding-3-small	~63.6%	\$1.95 cost (corpus embed); fast generation (small model).
Embedding	text-embedding-3-large	~66.9%	\$9 cost (corpus embed); improved recall; slower index build.
Embedding	all-MiniLM-L12-v2	~39.0%*	No cost; local model with good balance of speed & accuracy.
Embedding	Voyage-large-2	~62.0%	Internal large embed; performance similar to 3-small.
Embedding	infogrid Stella-base-v2	~39.8%	Open-source embed; lower accuracy, possible domain mismatch.
Re-ranker	**Voyage 2.5 lite**	**~57.5%**	Best accuracy vs latency; lightweight cross-encoder[23].
Re-ranker	Voyage 2.5 (full)	~56.2%	Slightly larger model; no accuracy gain over lite variant.
Re-ranker	Cohere rerank v3	~45.3%	Significant boost over none; ~10% slower than Voyage[23].
Re-ranker	BGE reranker (open-source)	~51.7%	Improves accuracy notably; fully local but less powerful.
Re-ranker	No re-ranking	~36.2%	Baseline retrieval only; many missed answers.
Inference	**Claude 4.0**	**~71.5%**	Highest accuracy; expensive (premium API); ~10–11 sec/Q.
Inference	**GPT-4 (OpenAI)**	**~69.8%**	Near-top accuracy; high cost; ~6–7 sec/Q latency.
Inference	GPT-4o-mini (≈13B model)	~52.0%	Moderate accuracy (up to 74% max); much lower cost; ~5–7 sec/Q.
Inference	DeepSeek-33B Chat	~58.0%	Decent accuracy for local model; no token cost; ~8.2 sec/Q.

Inference	DeepSeek Qwen-7B	~27.7%	Low accuracy; zero cost; very fast (~4 sec/Q); needs fine-tuning.
Inference	Mistral-7B (Instruct v0.3)	~24.4%	Similar to 7B above; slightly lower accuracy out-of-the-box.

Note: Lower accuracy for MiniLM is because it was evaluated mostly with 7B inference; with a stronger LLM, MiniLM's true potential would be higher (likely ~65% based on retrieval tests).

5. **Detailed Metrics**

To further illustrate the evaluation results, this section provides visual comparisons and additional metrics.

- Average accuracy by document **chunking + embedding** configuration. Each bar corresponds to a specific combination of chunking method (or model) and embedding model, averaged over all pipelines using that combo. We observe a clear separation between pipelines that used **LLM-based chunking** (top bars) and those that used only small/local chunking (bottom bars). Using an LLM (Claude or GPT-4) to preprocess documents (e.g. via semantic sectioning or summarization) combined with a powerful embedding (TE3-large) yielded the highest accuracy segments (72–76% range). The best combo was Claude 4.0 chunking with the large embedding, resulting in ~71.5% average accuracy. In contrast, pipelines using only the small **DeepSeek 7B** or **Mistral 7B** for chunking (with MiniLM or Stella embeddings) cluster at the bottom with ~27–35% accuracy. Notably, chunking with the intermediate **DeepSeek-33B** model plus large embedding reached ~58%, outperforming all 7B-based approaches but still below GPT-4 chunking. This implies that advanced chunking (semantic sectioning, etc.) with a strong model can boost retrieval and that gap widens with more capable models. Also, the choice of embedding within each chunking category has a visible effect (e.g., for GPT-4o-mini chunking, TE3-large > TE3-small > MiniLM, each step down reducing accuracy slightly).

- Average accuracy by **re-ranking model**. This bar chart highlights the benefit of adding a re-ranker. Without any re-ranking, the pipeline achieved only ~36% accuracy on average (far left). Using a **Cohere v3** re-ranker improved that to ~45%, and an open-source **BGE** re-ranker to ~52%. The **Voyage 2.5** models delivered the highest accuracy: ~56% for the full 2.5B model and ~57.5% for the distilled "lite" model. We see that Voyage-2.5-lite slightly edged out its larger counterpart, reflecting our findings that the distilled model was both faster and at least as effective. The general trend underscores that **any learned re-ranker is better than none**, and investing in a high-quality re-ranker (like Voyage or Cohere) yields substantial accuracy gains. In practice, the ~21-point jump from 36% (NoReranker) to 57.5% (Voyage2.5-lite) can be the difference between an unacceptable answer rate and a useful system. The latency overhead for these re-rankers was relatively minor (Voyage adds only a few hundred milliseconds per query), making re-ranking a very cost-effective step for improving accuracy.

- Average accuracy by **inference (answering) model** used. This compares the performance of different LLMs/SLMs when used as the final stage in the pipeline (with

various retrieval setups). **Claude 4.0** tops the chart at ~71.5% average accuracy, marginally higher than **GPT-4** (69.8%). Our internal **DeepSeek-33B** model reached 58%, significantly outperforming the smaller models. The **GPT-4o-mini** (approx 13B) model averaged ~52% – this average spans both high-performing runs (with advanced retrieval) and low-performing runs (with basic retrieval), indicating its performance is more dependent on the quality of retrieval. The bottom of the chart shows the **Qwen-7B** model at ~27.6% – reflective of its limited capabilities. This visual reinforces the earlier point: larger models are markedly more effective for this financial QA task. Another insight is the large gap between 13B and 33B local models (52% → 58%): scaling the model size roughly doubled the number of correct answers for local models. We expect this trend would continue with a 65B or 70B model (likely pushing local accuracy into the 60s). In operational terms, choosing the right inference model means balancing this accuracy curve against cost and speed: e.g., Claude and GPT-4 give the best results but incur high cost per query, whereas a mid-size model, if one can tolerate ~15–20 points lower accuracy, could handle queries at a fraction of the cost.

- Finally, to quantify latency and cost: the **LLM pipelines** (Claude/GPT-4) had an average per-query latency of ~10 seconds and an estimated cost of \$0.50–\$1.00 per query (given FinanceBench query + context lengths), whereas the **SLM pipelines** (7B–13B models) ran in ~4–6 seconds per query with essentially zero token cost (only infrastructure cost). This **latency gap** may narrow with optimization and batching, but it was consistently observed. For throughput-intensive settings, the faster response of smaller models can be a benefit, provided the lower accuracy is acceptable or mitigated by human oversight.

6. Conclusions & Recommendations

In conclusion, the dsRAG pipelines demonstrate strong capability on a challenging financial QA benchmark, especially when configured with powerful language models and robust retrieval components. The best pipeline (Claude-based) achieved high accuracy (76%), showing that with the right context, modern LLMs can answer detailed financial questions grounded in documents. However, there remains a considerable accuracy gap between large proprietary models and smaller open models. Deployment decisions must account for the accuracy-cost trade-offs, and there are several considerations and future enhancements we recommend to improve real-world performance and adoption:

Caveats & Considerations for Deployment

- **Data Sensitivity & Local Deployment:** In environments where data confidentiality is critical (e.g. handling proprietary financial documents), using external APIs like OpenAI or Anthropic may be infeasible. Our evaluation shows that a fully local stack is possible (using open models for embedding, re-ranking, and a 7B–33B model for inference), but one must **accept a significant accuracy trade-off** (dropping from ~75% to ~45% or below). For instance, the DeepSeek Qwen-7B pipeline reached only ~45% accuracy with a 13B assist, or ~33% with purely 7B, highlighting the challenge. It's important to set proper expectations in these cases: such a system may answer only half of the queries correctly, and will likely struggle with complex analyses. **Human oversight or verification** might be required for important queries when using an SLM pipeline.

- **Cost-Sensitive Scenarios:** Not all use cases justify the expense of GPT-4 or Claude. For cost-sensitive deployments, one might use a mid-tier model (like GPT-3.5 or a fine-tuned 13B) to get reasonably good accuracy (~60-70% on easier subsets) at a small fraction of the cost. The results suggest that, with a strong retrieval setup, a 13B model can sometimes approach the performance of a 100B model on retrieval-centric tasks. **Mixing model sizes** is a practical strategy: e.g., route straightforward queries to a cheap local model, and escalate only the complex queries to an expensive LLM. This keeps average cost per query low while maintaining quality on the tough question.
- **Fine-tuning and Domain Adaptation:** One way to improve smaller models is through fine-tuning. We recommend fine-tuning the 7B–13B models on domain-specific Q&A pairs and financial texts. This could boost their comprehension of balance sheets, financial terminology, and required calculations, narrowing the gap with larger model. Even GPT-3.5/4 can be domain-tuned (via techniques like retrieval augmentation or instruction tuning) to improve accuracy on finance questions. Such fine-tuning tends to yield better calibration and reduces irrelevant outputs. Similarly, embedding models and re-rankers can be fine-tuned on in-domain data to improve retrieval precision.
- **Model Distillation:** Organizations should consider a **distillation approach** where the knowledge and reasoning steps of a top LLM (e.g. GPT-4) are distilled into a smaller model. The smaller model is trained to reproduce the answers or rationale of the large model on a large set of finance QA examples. This could significantly improve the small model's accuracy without needing full scale. Our "DeepSeek-R1-Distill Qwen-7B" is an attempt in this direction – further iterations of distillation could yield better performing SLMs (possibly pushing the 7B model into the 50%+ accuracy range)[25].

Future Enhancements

- **Improving Relevant Segment Extraction (RSE):** Our analysis indicates that about **15% of errors** were due to retrieval misses – cases where the answer was present in the corpus but the system did not retrieve the correct segment in the top-k. The RSE mechanism (which groups related chunks into segments) is designed to increase recall[27][28], and it did help, but there were limitations. Sometimes RSE merged too many extraneous details, diluting the focus, or it failed to include a needed chunk due to strict grouping rules. Future work should refine the RSE logic, for example by using the query context to dynamically decide segment boundaries or by ensuring that at least one segment directly addresses each facet of a complex query. This can reduce retrieval errors and ensure the LLM always sees the critical evidence.
- **Hybrid Retrieval Strategies:** We recommend incorporating a hybrid of **vector search and keyword search** for FinanceBench-type queries. Pure vector search can miss specific numeric or name-based matches that a keyword lookup would catch. A hybrid approach (e.g. BM25 + dense retrieval) could improve recall, especially for factual lookup questions. The system could union the results or use an ensemble ranker to blend lexical and semantic results. This addresses cases where a relevant paragraph might not be semantically similar enough to the question embedding but contains obvious keyword overlaps (e.g., exact figures or company names).

- **Multi-hop Retrieval & Reasoning:** Some FinanceBench questions require combining information from multiple parts of a document or even across documents (multi-hop reasoning). Currently, our pipeline retrieves top chunks and gives them all at once to the LLM. A more advanced approach would be an iterative retrieval: the LLM answers partially or identifies a clue, which triggers a follow-up retrieval for the next piece of information, and so on. This **multi-hop retrieval** process can systematically handle questions that require a series of lookup steps. Implementing a multi-hop chain (possibly guided by the LLM generating follow-up queries) could improve accuracy on the hardest questions.
- **Larger Local Models:** The 33B model's success suggests that using a larger local model (e.g. 65B parameter model like Llama 2 70B) might further close the gap to GPT-4. Future evaluations should explore the performance of a fine-tuned 60–70B model in this pipeline. If a ~70B model could reach, say, 65%+ accuracy, it would present a very attractive private solution. The main barrier is computational: running such a model is resource-intensive, but with optimizations (4-bit quantization, faster transformer kernels) it is becoming feasible. We anticipate that as open models improve, the **gap between SLM and LLM will steadily shrink**.
- **Enhanced Fine-tuning of Components:** We touched on fine-tuning for the models; additionally, fine-tuning the **retrieval components** (embeddings and re-ranker) on the specific style of questions and documents in FinanceBench could yield better synergy. For example, training the embedding model on financial text embeddings (or using a financial domain embedding model) might improve retrieval of numeric tables or footnotes. Likewise, a re-ranker fine-tuned on FinanceBench QA pairs would learn to prioritize segments that likely contain answers (perhaps by recognizing patterns like "USD million" or specific financial ratios in text). These enhancements would make the pipeline more **domain-adapted**, improving accuracy without needing to scale up model size.
- **User Feedback Loop:** In practical deployment, incorporating a feedback loop where incorrect or low-confidence answers are flagged and the system learns from them can greatly boost performance over time. Techniques such as active learning (augmenting the training data with misanswered questions and their correct answers) can help address the specific failure modes of the model. Over the FinanceBench evaluation, we noticed certain systematic errors (e.g., confusing similar financial metrics, or format mistakes in numeric answers). These are ripe for correction via targeted fine-tuning or rule-based post-processing. Building such feedback into dsRAG will make it more robust and trustworthy for financial analysts.

In summary, **dsRAG** on the FinanceBench benchmark shows the promise of a retrieval-augmented approach for domain-specific QA. By carefully selecting and configuring pipeline components, we achieved strong results, and by applying the above enhancements, we expect further significant improvements. Deployers should weigh the accuracy needs against cost and choose an appropriate pipeline variant, keeping in mind the caveats discussed. With continuous model advances and domain adaptation, even cost-effective local models are likely to reach the performance needed for practical use in financial research and analysis, fulfilling the goal of grounded, **explainable Q&A** over enterprise data.

Helpful links:

dsRAG: https://github.com/D-Star-AI/dsRAG

FinanceBench: GitHub - patronus-ai/financebench

Link to dsRAG Evaluation using FinanceBench Results Spreadsheet:

Link to FinanceBench QA Evaluation using Evidence Text for all 150 QAs

Link to Examples of QAs with different categories such as Benchmark errors and so on

Link to Presentation and Demo