Executive Summary

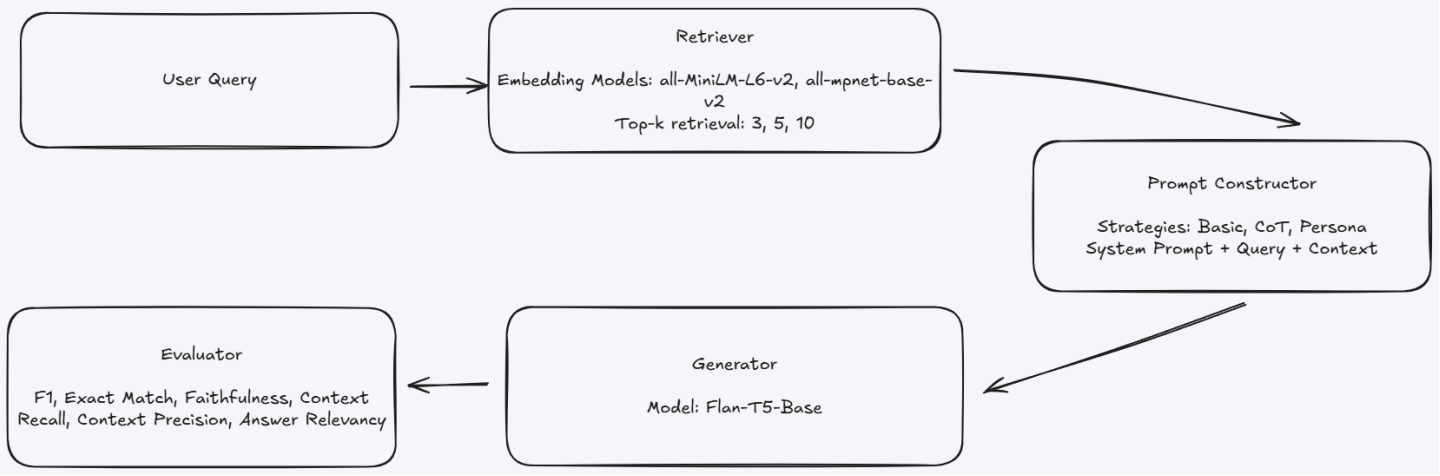
This project explored the design, development, and evaluation of Retrieval Augmented Generation systems, going from a naive baseline to an enhanced and improved architecture. The naive pipeline demonstrated that even a naive architecture can achieve good performance, with the optimal configuration (all-MiniLM-L6-v2, basic prompt, top-k=10) achieving an F1 score of 50.24, faithfulness of 0.92, and answer relevancy of 0.80.

The advanced pipeline incorporated query rewriting and context reranking, using a fetch-10/select-5 strategy to balance precision and recall. While these enhancements created a modular, production-ready system, evaluation accuracy metrics degraded due to aggressive query rewriting, which stripped semantic nuance, especially in yes/no questions. Despite that, faithfulness (0.78) and context recall (0.70) remained strong, which highlighted the robustness of the architecture even with semantic distortion.

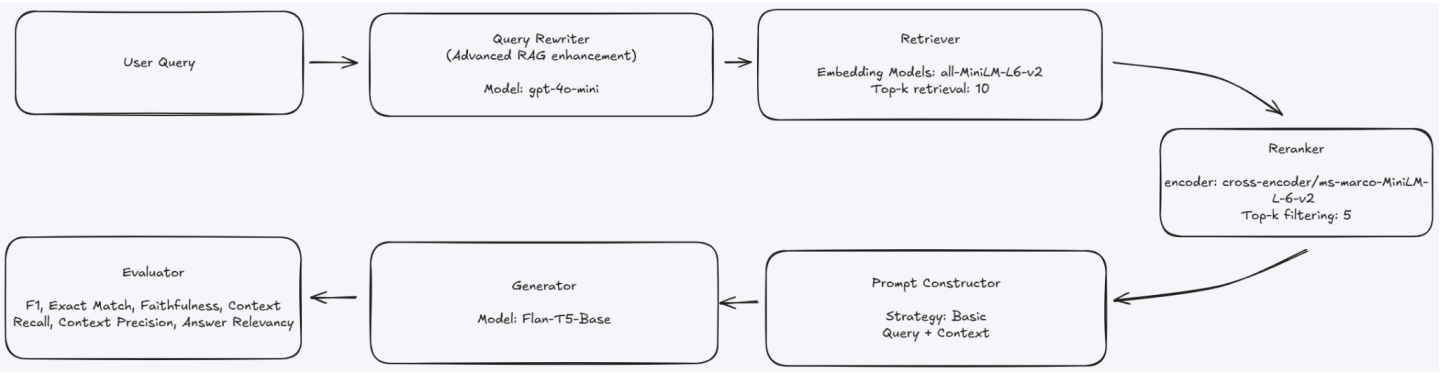
Overall, the project showed both the pros and considerations of RAG enhancements. The naive systems may outperform advanced systems on raw metrics, but the advanced pipelines offer greater scalability and robustness if refined appropriately. These findings underscore the importance of balancing retrieval coverage, rewriting fidelity, and answer consistency for deployment-ready RAG systems.

System Architecture

The naive pipeline followed the fundamental RAG flow. User queries were converted into embeddings (all-MiniLM-L6-v2 or all-mpnet-base-v2) and used to retrieve the top-k candidate passages. A prompt constructor incorporated the selected contexts into one of three prompt formats (basic, persona, chain-of-thought), which were then passed to Flan-T5-base for answer generation. Evaluation combined the HF SQuAD v4 metric (F1 score and Exact Match) with the RAGAS frameworks to measure faithfulness, context recall, context precision, and answer relevancy. This architecture was simple yet effective with competitive results



The advanced RAG pipeline introduced two features. First, a query rewriting function that reformulated user input to better align with the retriever's embedding space, aiming to improve recall. Second, a reranker that evaluated the top-10 retrieved contexts and selected the top-5 most relevant passages for inclusion in the prompt. This strategy balanced coverage with efficiency, and reduced context dilution. Due to the modular pipeline, the integrations were straightforward



Experimental Results

I started by taking a close look at the Naive RAG pipeline and testing it across three main dimensions: the embedding model, the prompting strategy, and the number of contexts retrieved (k). The idea was to see how each piece of the setup influenced overall performance and to understand the trade-offs we'd face between accuracy, recall, faithfulness, and efficiency. For embeddings, I compared: 'all-MiniLM-L6-v2', which is compact and efficient, and 'all-mpnet-base-v2', which is larger and heavier but sometimes better at capturing nuance. On the prompting side, I tried three different approaches: a straightforward basic prompt that included just the context and the question, a persona-style prompt that framed the model in a specific role, and a chain-of-thought prompt that encouraged step-by-step reasoning. Finally, I varied the retrieval depth, testing k=3, 5, and 10, to see how pulling in fewer versus more passages would affect the final answers.

Looking at all the combinations gave a clear sense of where the pipeline performed best and where it started to break down. For example, increasing k often helped recall but sometimes hurt precision, and that extra instruction in the prompt didn't always make the model more accurate. This baseline testing was important because it not only highlighted the best-performing setup but also showed the limits of the naive approach.

With that in hand, I moved on to the next stage of building the Advanced RAG pipeline. Here, I kept the best naive configuration fixed and added two enhancements:

- Query Rewriting to better match queries with the retriever
- Reranking to filter and prioritize the most relevant contexts. I also adopted a fetch-10/select-5 strategy to balance recall and precision.

Naive RAG Evaluation

Embedding Model	Prompt	# Top k	# F1 Score	# Exact Match	# Faithfulness	# Context Recall	# Context Precision	# Answer Relevancy
all_MiniLM_L6_v2	basic	10	50.24	39.65	0.92	0.84	nan	0.80
all_MiniLM_L6_v2	basic	3	47.48	37.69	0.76	0.61	0.72	0.71
all_MiniLM_L6_v2	basic	5	49.47	39.32	0.97	0.73	0.92	0.79
all_MiniLM_L6_v2	cot	10	42.43	31.48	0.74	0.83	nan	0.73
all_MiniLM_L6_v2	cot	3	37.57	27.12	0.88	0.57	0.83	0.63
all_MiniLM_L6_v2	cot	5	39.37	28.76	0.60	0.56	0.24	0.54
all_MiniLM_L6_v2	persona	10	44.91	33.66	0.78	0.73	nan	0.66
all_MiniLM_L6_v2	persona	3	41.96	31.37	0.83	0.67	0.79	0.74
all_MiniLM_L6_v2	persona	5	43.41	32.35	0.96	0.65	0.76	0.76
all_mpnet_base_v2	basic	10	48.44	38.24	0.65	0.55	nan	0.67
all_mpnet_base_v2	basic	3	48.60	38.78	0.60	0.65	0.83	0.76
all_mpnet_base_v2	basic	5	48.44	38.67	0.55	0.67	0.33	0.72
all_mpnet_base_v2	cot	10	42.19	31.48	0.85	0.78	nan	0.61
all_mpnet_base_v2	cot	3	39.72	29.74	0.74	0.55	1.00	0.50
all_mpnet_base_v2	cot	5	39.63	29.41	0.77	0.68	0.69	0.61
all_mpnet_base_v2	persona	10	44.59	33.44	0.73	0.68	nan	0.64
all_mpnet_base_v2	persona	3	42.69	32.68	0.90	0.72	0.93	0.79
all_mpnet_base_v2	persona	5	43.06	32.35	0.86	0.60	0.84	0.74

Embeddings

Between the two tested embedding models, all-MiniLM-L6-v2 and all-mpnet-base-v2, the results for MiniLM were much stronger. With MiniLM, F1 scores consistently broke into the high 40s and low 50s, while MPNet typically stayed several points lower. For example, with a basic prompt at k=10, MiniLM achieved an F1 of 50.24 compared to MPNet's 48.44. Although both models delivered competitive recall, MiniLM produced more faithful and relevant answers on average.

Prompting Strategies

Prompt design also strongly influenced outcomes. The basic prompt was the most reliable, yielding the highest F1 and exact match values across both embeddings. Persona prompts occasionally improved faithfulness but at the cost of raw accuracy, with F1 scores slipping into the low 40s. Chain-of-thought prompts were the least effective here: although they sometimes improved reasoning traces, they consistently lowered accuracy, with F1 scores around 37 - 42. In short, more complex prompting strategies did not translate into stronger performance with this relatively small model.

Since our generator was 'Flan-T5-base', a relatively small model, it did not have the reasoning capacity to take advantage of elaborate instructions. This explains why the basic prompt was the most reliable. If we have used a larger, more capable model like 'Flan-T5-XL' or a GPT-class model, the outcome would have been very different, as the bigger models would be good at following complex reasoning and persona conditioning

Top-K Retrieval

Retrieval depth showed a similar trade-off. With k=3, precision was stronger (≈0.72) but recall dropped. Increasing to k=5 balanced the two, but k=10 gave the highest recall (0.84) and the best F1 score (50.24). However, the precision metric sometimes collapsed to NaN at k=10, indicating that broader retrieval

occasionally diluted the context set with irrelevant passages. This highlights a production concern, ie, wider retrieval increases recall but may harm downstream answer quality unless paired with filtering.

This pattern reflects an important consideration for retrieval-augmented systems: more context isn't always better. While a larger k increases the chance that the right evidence is somewhere in the context, it also increases the risk of diluting the context window with irrelevant or even misleading passages. Smaller models like Flan-T5-base are especially vulnerable to this problem because they lack the reasoning capacity to filter out noise on their own. A larger model might handle a wider context window more gracefully, but even then, precision would still degrade without a reranking mechanism.

Best Naive Setup

Combining these observations, the best naive pipeline used all-MiniLM-L6-v2 with a basic prompt and k=10. This combination worked well for several reasons. MiniLM embeddings consistently delivered stronger semantic matches than MPNet, which meant that the retrieved passages were more aligned with the original query. Pairing that with a basic prompt avoided overloading Flan-T5-base with unnecessary instructions and gave it the best chance to extract a direct answer. Finally, using k=10 maximized recall and ensured that the model had enough supporting evidence to draw from, which in turn boosted F1 and exact match

Advanced RAG Evaluation


```JSON
"metrics": {
    "f1_score": 22.17080807447771,
    "exact_match": 14.4880174291939,
    "faithfulness": 0.78125,
    "context_recall": 0.7,
    "context_precision": null,
    "answer_relevancy": 0.5592362388643247
  }
```


Building on the naive baseline, the advanced pipeline introduced two enhancements designed to address the weaknesses we observed earlier:

Query Rewriting

Query rewriting was added to tackle the mismatch between user queries and the retriever's indexing. In the naive setup, queries were passed through as-is, which worked reasonably well but sometimes failed when the query contained stop words, grammatical padding, or phrasing that didn't align well with the embedding model's training data. Rewriting aimed to normalize and simplify these queries so the retriever could better match them to relevant passages, improving recall.

Reranking

Reranking was introduced to directly address the precision problems we saw at k=10 in the naive pipeline. By initially retrieving a broader pool of 10 passages and then re-scoring them with a cross-encoder, the goal was to keep the benefits of wide recall while filtering out irrelevant or low-quality contexts. This was also where the fetch-10/select-5 strategy came from: cast a wide net, then trim to the most useful subset.

For reranking, we selected the cross-encoder ms-marco-MiniLM-L6-v2. This model was a practical choice because it's lightweight enough to be efficient in a pipeline while still being specifically fine-tuned on the MS MARCO dataset, a large-scale passage ranking benchmark. In other words, it was designed for exactly this task: scoring query-passage relevance. Heavier cross-encoders (like mpnet-base variants) could potentially achieve higher raw accuracy, but they come with significant latency and compute overhead. The MiniLM reranker provided a good balance between performance and scalability, making it a natural fit for a system that we wanted to keep production-ready.

Result

In theory, these two advancements should have fixed the precision issues seen in the naive setup. In practice, however, the results were mixed. Faithfulness (0.78) and context recall (0.70) remained robust, showing that the pipeline still surfaced meaningful evidence. But F1 and exact match dropped sharply (22.17 and 14.49, respectively).

This was due to the rewriting module. It aggressively strips stop words and grammatical cues, which sometimes distorts the intent of the original query. For instance, the yes/no question “Was Abraham Lincoln the sixteenth President of the United States?” was reduced to “Abraham Lincoln sixteenth President United States”, which confused the generator and led to malformed answers.

The reranker itself did its job of fetching 10, then filtering down to 5 improved the input set’s overall quality, but it couldn’t recover from the semantic loss introduced during rewriting.

As a result, while the advanced pipeline was far more modular and production-friendly, its raw accuracy lagged behind the naive baseline.

Production Considerations:

Both pipelines are highly modular, with helper functions abstracting retrieval, prompting, generation, and evaluation. This design simplifies conversion into production scripts or microservices. For deployment, the retriever can scale via vector databases like FAISS or Milvus, and reranking can run in parallel to handle large candidate pools.

Scalability trade-offs are clear: larger k improves recall but increases latency and context dilution. The fetch-10/select-5 strategy represents a practical compromise. To further improve efficiency, caching embeddings and batching retrieval calls can reduce overhead.

Deployment on cloud-native platforms would support horizontal scaling. The pipeline is container-friendly and could be integrated into CI/CD workflows.

Monitoring is essential: metrics like faithfulness and answer relevancy should be tracked in real-time, as they better capture user-facing reliability than F1 or EM alone.

The small language models (Flan-T5-base) restricting generation quality and rewriting sophistication is one of the limitations. More advanced LLMs could preserve semantic integrity during rewriting while improving answer fluency. Additionally, reranking relies heavily on embedding quality, making hybrid retrieval (dense + sparse) a promising direction.

While naive RAG may be preferable for immediate accuracy, advanced RAG offers a future-proofed architecture. With refinements in query rewriting and larger generation models, the enhanced system would be ready for real-world deployment at scale.

Appendix

AI Usage:

1. VS Code Copilot with GPT-5 mini and Gemini for Tab Completion of Code
2. Grammarly for fixing typos in the report
3. ChatGPT
    a. Script generation to create CSV - Link
    b. Understanding RAGAS vs ARES - Link
    c. System Prompt Generations - Link
    d. Understanding Milvus - Link
4. Claude
    a. Issues with Milvus - Link
    b. Understanding RAGAS metrics - Link

Along with this, the documentation for all the different packages, like Ragas, and SentenceTransformers, was referred to learn more about them, see example usage and syntax