Executive Summary

This report documents the phased development, evaluation, and enhancement of a Retrieval-Augmented Generation (RAG) system, from a baseline prototype to a production-ready pipeline.
The initial phase focused on optimizing a baseline RAG pipeline by comparing two embedding models (all-MiniLM-L6-v2 with 384 dimensions and all-mpnet-base-v2 with 768 dimensions) and three distinct prompting strategies. Key findings indicate that the larger 768d embedding model significantly outperforms the 384d model, achieving a peak F1 score of 81% versus 77%. A "Persona" prompt proved most effective for the generator, while a "Chain-of-Thought" approach consistently degraded performance.

The second phase introduced an advanced RAG pipeline incorporating query rewriting and a cross-encoder reranking stage. Contrary to expectations, this enhanced system underperformed the optimized baseline, with the average F1 score decreasing from 73.6% to 69.6%. This highlights that adding complexity does not guarantee improvement and requires careful tuning. The optimized baseline system demonstrates strong foundational performance.

System Architecture

1. Data Ingestion and Indexing:
   The pipeline begins with processing text documents. Passages are loaded and then segmented into smaller, manageable chunks using the RecursiveCharacterTextSplitter, configured with a chunk size of 512 characters and an overlap of 50. This chunking strategy is a trade-off, designed to maintain semantic context within each chunk while keeping them small enough for efficient processing by the embedding model.

2. Embedding and Vector Storage:
   Each text chunk is converted into a high-dimensional vector using a sentence-transformer model. Two models were evaluated: all-MiniLM-L6-v2 (384 dimensions) for efficiency and the larger all-mpnet-base-v2 (768 dimensions) for potentially higher accuracy. These embeddings are indexed and stored in a Milvus Lite vector database, which uses the L2 distance metric for similarity search.

3. Retrieval and Generation Core:
   When a query is received, it is first embedded using the same model as the documents. The system then executes a similarity search in Milvus to retrieve the top-k most relevant passage chunks. These retrieved chunks are concatenated and formatted into a single context string. This context, along with the original query, is passed to the generative model, google/flan-t5-base. The design decision to test multiple prompt strategies (Instruction, Persona, Chain-of-Thought) was made to assess the generator's sensitivity to input framing. The final answer is generated based on the provided context and prompt.

   <More details in Architecture Document>

Experimental Results

Experiment 1: Embedding Model and Dimension Comparison
The first experiment compared the performance of the 384-dimension all-MiniLM-L6-v2 model against the 768-dimension all-mpnet-base-v2 model.
The 768d model demonstrated consistently superior performance across all prompt strategies and top-k retrieval settings. The highest F1 score achieved by the 768d model was 81.0% (using the "Persona" prompt with k=3), compared to a peak of 77.0% for the 384d model under similar conditions.
Likewise, the 768d model achieved a top Exact Match score of 72%, while the 384d model peaked at 68%. This result confirms that the greater representational capacity of the larger embedding model provides a significant advantage in retrieval accuracy for this task.

Experiment 2: Prompt Strategy Analysis
Three prompt strategies were evaluated: a basic "Instruction" prompt, a "Persona" prompt instructing the model to act as an expert encyclopedia, and a "Chain-of-Thought" (CoT) prompt. The results showed a stark difference in effectiveness.

- Persona Prompt: Consistently the top performer for both embedding models, indicating that providing role-based context helps the flan-t5-base model generate more accurate and well-formatted answers.
- Instruction Prompt: A reliable second-place performer, providing a solid baseline.
- Chain-of-Thought Prompt: This strategy failed across all tests, yielding an Exact Match score of 0% and F1 scores typically between 10-15%.

Experiment 3: Impact of Top-K
The number of retrieved documents (top_k) was varied between 1, 3, and 5. For the effective "Persona" and "Instruction" prompts, increasing k from 1 to 3 generally boosted performance by providing more context. The leap from k=3 to k=5 offered marginal or sometimes even negative performance enhancements, suggesting, top_k = 3 is perfect for this dataset.

Experiment 4: Advanced RAG Techniques
A final experiment compared the optimized baseline (384d model, k=3, Persona prompt) against an advanced pipeline featuring query rewriting and cross-encoder reranking. Unexpectedly, the advanced system's performance declined. The baseline achieved a 64% Exact Match and 73.6% average F1, while the advanced system scored 60% and 69.6%, respectively.

Enhancement Analysis

Two advanced techniques were integrated into the pipeline: Query Rewriting and Cross-Encoder Reranking.

1. Query Rewriting
   This enhancement uses the google/flan-t5-base model to generate two alternative phrasings of the original user query. The rationale is that different phrasings might better match the language used in the source documents, thus improving the chances of retrieving relevant information. Documents were retrieved for the original query and both variants to create a larger, more diverse pool of candidate passages.

2. Cross-Encoder Reranking
   After an initial, broad retrieval (fetching 10 passages for each query variant), a more sophisticated reranking step was applied. Unlike the bi-encoder used for initial retrieval (which embeds the query and document separately), a cross-encoder (cross-encoder/ms-marco-MiniLM-L-6-v2) processes the query and each candidate passage together. This allows for deeper semantic analysis and a more accurate relevance score. The top 3 most relevant passages after reranking were selected for the final context.

Production Consideration

Scalability: The current implementation relies on Milvus Lite, which runs locally and is not suitable for production workloads. The first step would be to migrate the vector database to a distributed, scalable solution like a full Milvus cluster or a managed vector database service like Weaviate or ChromaDB Cloud. This would support a much larger document corpus and handle concurrent user requests. While flan-t5-base is a relatively small LLM, the infrastructure for hosting it must be designed to handle the desired requests-per-second, potentially requiring GPU acceleration and auto-scaling inference endpoints.

Deployment Recommendations: Based on the experimental results, the most reliable and cost-effective candidate for an initial deployment is the baseline RAG pipeline using the all-mpnet-base-v2 (768d) embedding model, the "Persona" prompt strategy, and a top_k of 3. This configuration provided the highest accuracy without the added latency and complexity of the advanced techniques. The advanced pipeline (query rewrite and rerank) adds two significant latency sources (an extra LLM call and the cross-encoder computation) and should not be deployed until it can be tuned to demonstrate a clear performance benefit.

Limitations: The system's performance has only been validated on a small set of questions. Further testing on diverse, domain-specific datasets is crucial. The model is also sensitive to prompt structure, as shown by the failure of the CoT strategy, necessitating robust prompt management and versioning.

Appendices

AI or LLMs have been used to correct the grammar and change the tone of the report. The LLMs were also used to have outputs present in the codebase in a format that is easily viewable for the viewer, ie, making the tables, summary from the outputs of the code etc.

The three Jupyter notebooks are submitted along with the report.

The other reports attached to this one have the other required information, particular to their Phase or workflow.

1. Domain Documents
2. Architecture Document
3. Naive RAG Implementation
4. Advanced RAG Implementation