Technical Report

Executive Summary

This report evaluates the progression from a naive to an enhanced Retrieval-Augmented Generation (RAG) pipeline developed on the RAG Mini Wikipedia dataset. The naive implementation employed basic vector retrieval and single-shot generation, whereas the enhanced pipeline incorporated context window optimization and semantic reranking for evidence selection.

Key findings reveal significant improvements in both objective metrics and qualitative answer quality after these enhancements. The enhanced RAG pipeline delivered a 21 percentage point improvement in “perfect answer” rate (simultaneously correct, faithful, and helpful) compared to the naive baseline, alongside absolute improvements exceeding 20% each in correctness, faithfulness, and helpfulness. These results were validated across 180 diverse queries evaluated using the RAGAS framework and GPT-4o mini LLM as a judge. The modular codebase, persistent ChromaDB storage, and well-structured configurations demonstrate robust deployment readiness for cloud and research scenarios. The work establishes both a reproducible benchmark and a practical framework for extending RAG capabilities in production settings.

Written by: Aditya Sannabhadti

System Architecture

The RAG system was engineered for extensibility, reproducibility, and efficiency, leveraging modular Python code and cloud-integrated storage. The data pipeline starts by preprocessing and chunking the RAG Mini Wikipedia documents into tokens (sizes: 256, 384, and 512 examined), with each passage tokenized to preserve sentence boundaries and semantic flows. The embedding model—Gemma 300M—encodes chunks as 300-dimensional vectors.

ChromaDB serves as the vector database, mounted on Google Drive for cloud persistence, ensuring consistent experiment reproduction and cross-platform flexibility (Google Colab, local). All passage vectors and their metadata are indexed using cosine distance, which supports both fast similarity search and later reranking procedures.

The baseline pipeline uses nearest-neighbor retrieval and direct user-query fusion with retrieved context. Persona prompting, chain-of-thought, and instruction prompting modes are supported, with persona prompting empirically found to perform best. Model inference is handled by Llama 3.1 1B, selected for accessible deployment and favorable compute/resource trade-offs.

In the enhanced pipeline, two major features are integrated: (1) context window optimization, producing focused, high-signal prompts using semantic density scoring within the LLM's input capacity, and (2) reranking, using MiniLM cross-encoder to reorder the top-k retrieved chunks by query relevance, feeding only the most relevant contexts to final generation. Configuration files enable flexible adjustment of all major pipeline settings such as chunk size, top_k, and reranking thresholds.

Automated evaluation and logging routines are built in, and a configuration-driven approach supports scalability, fast parameter sweeps, and extension to new evaluation or deployment environments.

Experimental Results

The evaluation, spanning 180 open-domain queries, highlights the clear capability gains of the enhanced RAG system over the baseline. RAGAS, with an LLM judge, scored every output in terms of correctness, faithfulness, and helpfulness, tagging as a “perfect answer” any response meeting all three criteria:

Metric	Naive RAG	Enhanced RAG
Correct	74 (41.1%)	113 (62.8%)
Faithful	80 (44.4%)	120 (66.7%)
Helpful	78 (43.3%)	118 (65.6%)
Perfect (All)	72 (40.0%)	110 (61.1%)

1. **Correctness:** Enhanced pipeline responses aligned with ground truth 62.8% of the time, a jump from 41.1% for the naive version. This highlights improved retrieval precision and generative accuracy.

2. **Faithfulness:** The rate of context-grounded (“hallucination-free”) answers moved from 44.4% to 66.7%, illustrating the benefit of more sophisticated, evidence-driven passage selection and filtering.

3. **Helpfulness:** Answers that provided practical information or satisfied intent rose from 43.3% to 65.6%, confirming that reranking and windowing help the LLM focus on user needs.

4. **Perfect Answers:** Over three-fifths (61.1%) of enhanced system answers were simultaneously correct, faithful, and helpful, as opposed to only two-fifths (40%) for the naive baseline.

No errors, technically broken outputs, or system failures were detected in either pipeline.

This uplift is directly attributable to architectural changes, especially reranking and context window selection. Side-by-side qualitative review confirmed that improved context not only reduced factual errors but steered the LLM to more complete, user-relevant answers.

Enhancement Analysis

Context Window Optimization:
Adaptive prompt construction selectively packs only the most relevant retrieved segments into the LLM context window. This increases “signal” and reduces LLM distraction, leading to more accurate and contextually appropriate output. The main challenge lay in harmonizing token count checks and overlap scoring, so important yet concise passages dominated the context.

Reranking (MiniLM):
A cross-encoder reranker—applied after dense vector retrieval—meaningfully increased the relevance of context fed to the generator. Top-scoring passages (based on direct query-passage semantic matching) were consistently more useful than strict embedding-similar neighbors. Trade-off: reranking introduced moderate latency, but resulted in fewer hallucinations and supported higher faithfulness and perfect answer rates.

Implementation Challenges:
Memory management (especially in Colab during batch reranking or bulk inference), keeping API secrets secure, and ensuring full pipeline reproducibility across cloud and local runs, all required careful code and config design.

Effectiveness:
Overall, each enhancement directly contributed to measurable performance lifts. Experimental controls (avoiding prompt or eval leakage, fixed seeds for embedding generation) assured that results reflect true system change rather than confounding factors.

Production Considerations

Scalability & Cloud Readiness:

- The modular, configuration-driven architecture is amenable to both research and production scaling.
- Using ChromaDB with persistent Drive storage ensures resilience and flexibility for distributed or cloud-based deployments.
- Codebase is designed for Google Colab, but can be adapted to other platforms or pipelines, and supports batch inference for larger evaluation and retraining cycles.

Deployment Recommendations:

- For frequent retraining or integration into API-based services, decoupling, reranking from retrieval and running as an on-demand microservice is recommended.
- Preloading common vectors and dynamic context window sizing will further reduce runtime cost and latency.

Limitations:

- While the system handled a moderate dataset and query set well, scaling to very large corpora may require more robust sharding or distributed vector storage.
- Reranking and advanced context selection, while beneficial for answer quality, do add computational overhead.
- Further alignment to enterprise and domain-specific use cases may require tuning or more specialized models.

Appendices

AI Usage Log

- **Tool:** OpenAI GPT-4o mini (RAGAS integration)
- **Purpose:** Automated LLM-based scoring for correctness, faithfulness, helpfulness
- **Input:** Batches of (question, predicted answer, ground truth)
- **Output Usage:** Score columns for downstream metrics, CSV logs
- **Verification:** Random human review on sample results, parity with manual scoring checks

Technical Specifications

- **Python 3.9+ or Google Colab with A100 GPU preferable**
- **Requirements:**
  In the Requirements.txt file
- **Configuration:**
  Provided config.txt (chunk length, top_k, embedding model, LLM model)

Reproducibility Instructions

1. Clone the project and install dependencies (`pip install -r requirements.txt`)
2. Mount Google Drive for persistent ChromaDB storage
3. Set your Hugging Face and OpenAI API keys as environment variables
4. Adjust pipeline parameters in config files as needed
5. Run provided notebooks/scripts for data preprocessing, retrieval, enhancement, and evaluation
6. Evaluate results—aggregated metrics are stored in CSV for reporting
7. Review/validate with both automated metrics and optional human spot checks