Technical Report Structure (Phase 5)

Executive Summary (200 words): Key findings, performance improvements, deployment readiness

System Architecture (300 words): Technical implementation details, design decisions, trade-offs

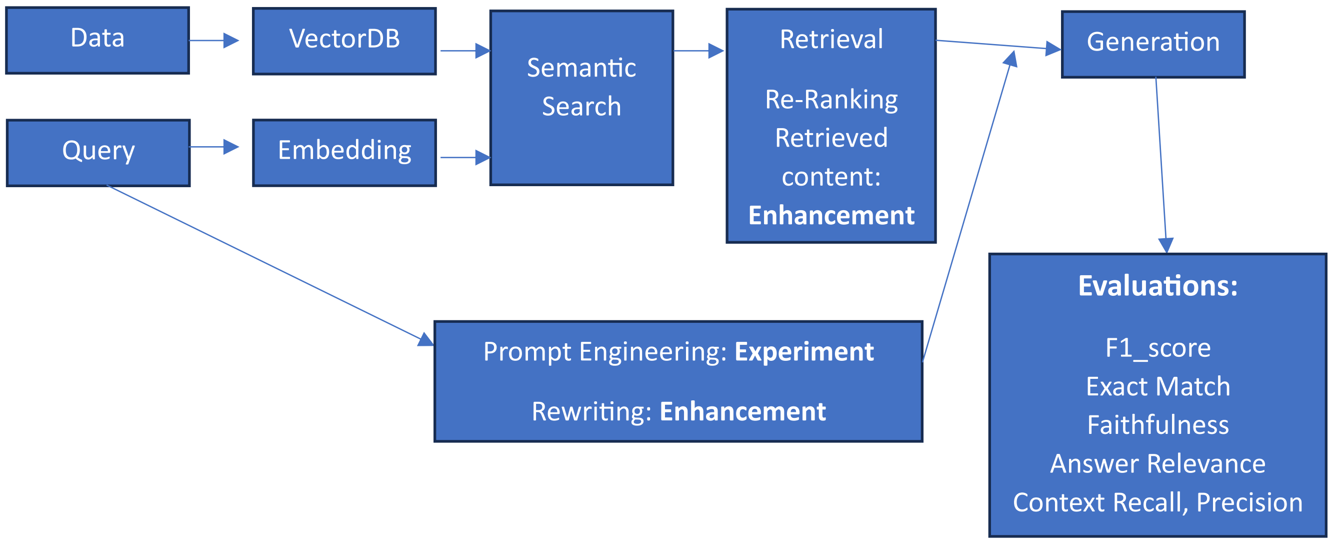
This exercise has followed the structure shown below. The development of the pipeline, as well as notebooks – followed by the evaluations were all done locally. New environments in python were created for the assignment itself as well as the RAGAs evaluations. This was done to ensure quick and clean runs, and while setup had hiccups – it wasn't messy. The decision to do this locally as well as model choices, evaluation tests was largely based in cpu capacity, and stay indifferent to session timeouts and cloud limits – considering this process was iterative. Except disappoint RAGAs evaluation runs, everything was smooth and worth the trade-off.

Configurations and requirements were laid out separately as is noted below – using these, pipelines were created. Using notebooks, pipelines were used and improved. The results were generated and saved separately using notebooks. Considering the production-like design expected, efforts were made to ensure the reproducibility of this exercise. A git workflow was initiated and adhered to.

For naïve RAG, sentence-transformers like all-MiniLM-L6-v2 and all-mpnet-base-v2 were used. The baseline is measured using all-mpnet-base-v2, and this was due to it's better performance upon initial exploration as has been documented in the evaluations. Later implementation of RAG (naïve and enhanced) used just the better all-mpnet-base-v2 model. While this wasn't super efficient considering local deployment, it was effective. FAISS was the vector DB used throughout the exercise. The inner product for cosine similarity with normalization was used for indexing in the Vector DB via FAISS. Retrieval was executed by semantic search with varying top_k iterations. The generation of answers post retrieval was done using Google's Flan T5-base model. All these models were retrieved from HuggingFace. The model choice was made due to it's open source nature, manageable size with local inference and zero API costs. Experimentation with embedding dimensions = 384, 768 and top_k retrieval were conducted as has been documented in the evaluations.

```
from-RAG-to-riches/
 ├── src/
|    ├── naive_rag.py
|    ├── enhanced_rag.py
|    ├── evaluation.py
|   └── utils.py
 ├── data/
|    ├── processed/
|   └── evaluation/
 ├── results/
|    ├── naive_results.json
|   ├── enhanced_results.json
|   └── comparison_analysis.csv
├── notebooks/
|   ├── data_exploration.ipynb
|   ├── system_evaluation.ipynb
|   └── final_analysis.ipynb
├── docs/
|   ├── setup_instructions.md
|   ├── evaluation_report.pdf
|   └── technical_appendix.md
└── requirements.txt
```

The exercise's architecture looks like this:


Data
VectorDB
Query
Embedding
Semantic Search
Retrieval
Re-Ranking Retrieved content: **Enhancement**
Generation
Prompt Engineering: **Experiment**
Rewriting: **Enhancement**
Evaluations:
F1_score
Exact Match
Faithfulness
Answer Relevance
Context Recall, Precision

Experimental Results (400-500 words): Comprehensive performance analysis with statistical support

Enhancement Analysis (300-400 words): Advanced technique effectiveness, implementation challenges

Production Considerations (200-300 words): Scalability, deployment recommendations, limitations

Appendices: Complete AI usage logs, technical specifications, reproducibility instructions