Technical Report

Retrieval-Augmented Generation (RAG) System for Dr. X's Data

Author: Mustafa Ahmad

Date: April 16, 2025

ABSTRACT

This report presents the design, implementation, and performance analysis of a Retrieval-Augmented Generation (RAG) system developed to process and analyze **Dr. X's** data. The system integrates document extraction, text chunking, embedding generation, vector storage via FAISS, and a local LLaMA model for natural language generation. The system supports multiple file formats (PDF, DOCX, CSV, Excel) and translation capabilities. Performance measures (extraction efficiency, RAG response time, translation speed) are evaluated, with graphs to visualize system competencies. Challenges, limitations, and possible future improvements are discussed to provide an overall view of the system's performance.

1. INTRODUCTION

The Retrieval-Augmented Generation (RAG) framework integrates information retrieval and natural language generation to generate contextually relevant and accurate answers to user questions. This project applies a RAG system to **Dr. X**'s data, which is kept in the "**Dr. X**" folder, to retrieve insights, summarize texts, and answer questions effectively. The system utilizes open-source tools such as Sentence Transformers for embeddings, FAISS for storing vectors, and a quantized LLaMA model for generation.

The objectives for this project include:

- To extract and process heterogeneous document formats.
- To construct a searchable vector database for efficient retrieval.
- To generate accurate and contextual responses using a local LLaMA model.
- To examine system performance through metrics and visualizations.

This report is organized as follows:

Section 2 discusses methodology.

Section 3 is implementation.

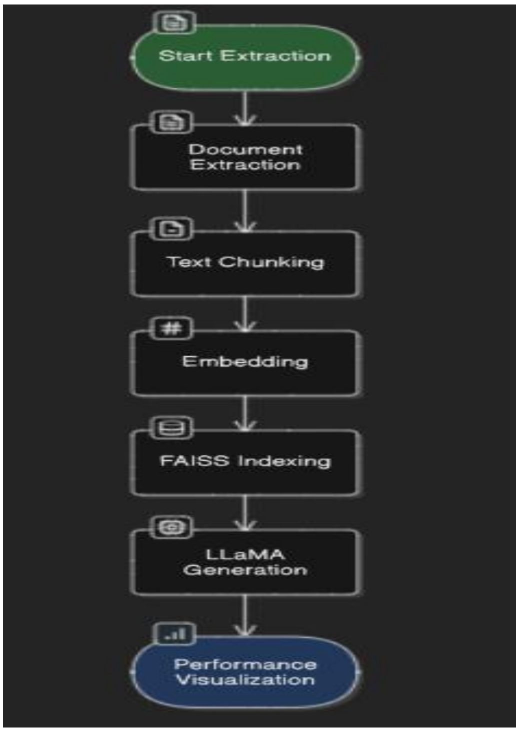
Section 4 is performance analysis.

Section 5 is challenges and limitations.

Section 6 concludes with future directions.

2. METHODOLOGY

The RAG system is designed in a modular pipeline manner, as illustrated below:



2.1 Document Extraction

The system processes different file formats (PDF, DOCX, CSV, Excel) using specialized libraries:

PyMuPDF (fitz) for PDF text extraction.

python-docx for DOCX.

pandas for CSV and Excel.

All files are opened, and their text content is drawn into a monolithic string format for processing.

2.2 Text Chunking

To accommodate processing of large documents, text is chunked into segments using the tiktoken tokenizer with cl100k_base encoding. The chunking process (chunk_text_with_overlap) ensures:

- A maximum of 512 tokens per chunk.
- An overlap of 50 tokens to preserve context between chunks.
- This is a trade-off between computational efficiency and contextual integrity.

2.3 Embedding Generation

Text chunks are embedded into dense vectors by the all-MiniLM-L6-v2 model of Sentence Transformers. This model produces 384-dimensional embeddings, fine-tuned for semantic similarity tasks.

2.4 Vector Storage and Retrieval

The FAISS library is employed to build a flat L2 index for chunk embedding storage. The index allows efficient similarity searches for retrieving the most similar chunks for a query.

2.5 Language Model Integration

A quantized LLaMA model (Llama-3.2-1B-Instruct-Q4_K_M-GGUF) is loaded for response generation. The model uses processed retrieved chunks as context to produce correct and coherent responses.

2.6 Performance Measurement

The system tracks:

Extraction efficiency: Number of chunks and time per file.

RAG performance: Response time and number of tokens for sample queries.

Translation performance: Tokens per second for Arabic translation.

Summary metrics: Number of tokens of generated summaries.

Visualizations are plotted using Matplotlib and Seaborn to display these metrics.

3. IMPLEMENTATION

The RAG system is implemented in a Jupyter Notebook (main.ipynb), with cells organized for reproducibility and modularity. Below is a step-by-step description of each part.

3.1 Environment Setup

The system begins by installing requirements in requirements.txt. Of interest are libraries for:

tiktoken for tokenization.

sentence_transformers for embeddings.


```
gperf
cmake
pkgconfig
sentencepiece
llama-cpp-python
sentence-transformers
faiss-cpu
tiktoken
python-docx
PyMuPDF
pytorch
pandas
openpyxl
rouge-score
transformers
langdetect
matplotlib
seaborn
reportlab
```

faiss-cpu for vector indexing,

pandas, matplotlib, seaborn for data processing and visualization.

3.2 Text Extraction and Chunking

The load_and_chunk_all_files function iterates over the "Dr.X" folder, extracting text from supported file formats. For every file:

Text is extracted using format-dependent methods (e.g., extract_text_from_pdf).

Chunks are created with overlap to ensure continuity.

Metadata (filename, chunk number) is preserved for traceability.

The cell reports statistics, which include chunks generated and elapsed time.

3.3 Embedding and Vector Database

The all-MiniLM-L6-v2 model generates embeddings for each chunk. The embeddings are stored in an instance of FAISS IndexFlatL2 for fast nearest-neighbor searching.

3.4 LLaMA Model Integration

The LLaMA model is instantiated with comprehensive metadata, checking its architecture (16 blocks, 2048 embedding dimension) and quantization (Q4_K, 762.81 MiB). The model accepts processed retrieved chunks to generate responses.

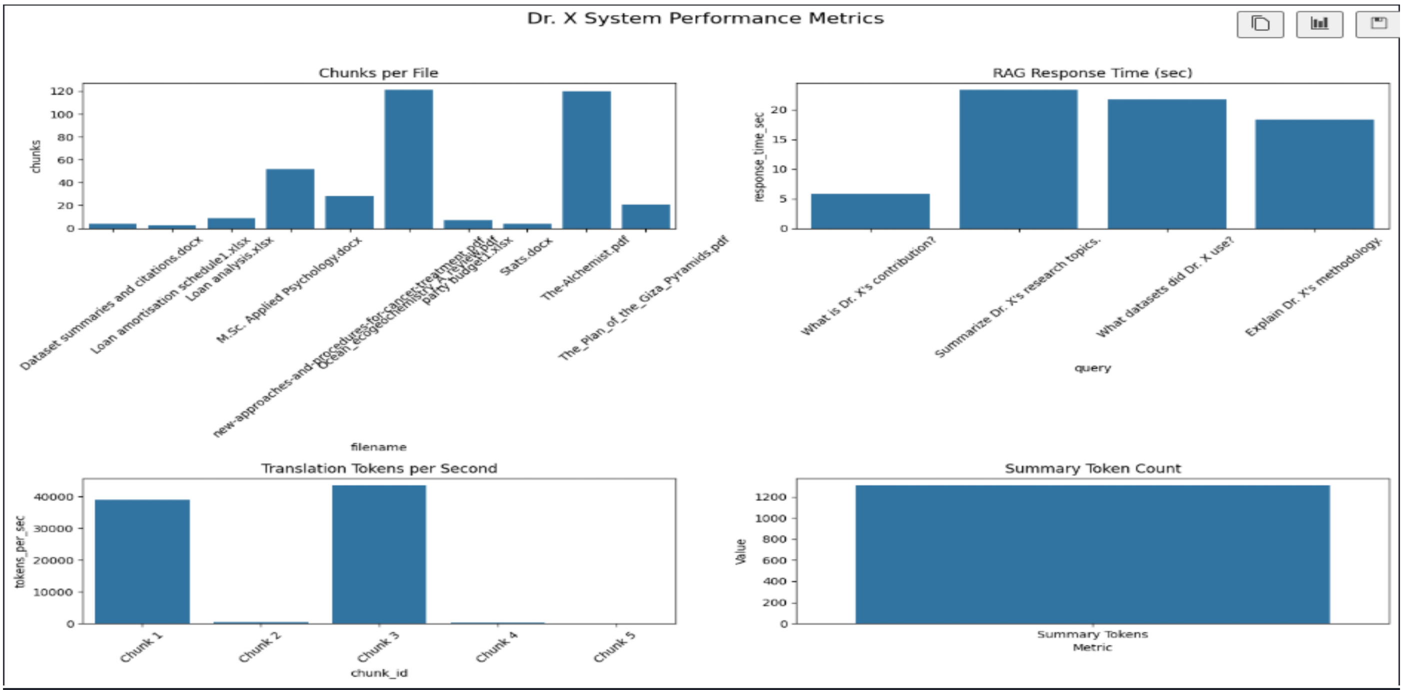
```
llama_model_loader: loaded meta data with 30 key-value pairs and 147 tensors from C:/mystuff/Project1/Llama-3.2-1B-Instruct-Q4_K_M-GGUF/
llama_model_loader: Dumping metadata keys/values. Note: KV overrides do not apply in this output.
llama_model_loader: - kv   0:                       general.architecture str              = llama
llama_model_loader: - kv   1:                               general.type str              = model
llama_model_loader: - kv   2:                               general.name str              = Llama 3.2 1B Instruct
llama_model_loader: - kv   3:                           general.finetune str              = Instruct
llama_model_loader: - kv   4:                           general.basename str              = Llama-3.2
llama_model_loader: - kv   5:                         general.size_label str              = 1B
llama_model_loader: - kv   6:                               general.tags arr[str,6]       = ["facebook", "meta", "pytorch", "llam...
llama_model_loader: - kv   7:                          general.languages arr[str,8]       = ["en", "de", "fr", "it", "pt", "hi", ...
llama_model_loader: - kv   8:                          llama.block_count u32              = 16
llama_model_loader: - kv   9:                       llama.context_length u32              = 131072
llama_model_loader: - kv  10:                     llama.embedding_length u32              = 2048
llama_model_loader: - kv  11:                  llama.feed_forward_length u32              = 8192
llama_model_loader: - kv  12:                 llama.attention.head_count u32              = 32
llama_model_loader: - kv  13:              llama.attention.head_count_kv u32              = 8
llama_model_loader: - kv  14:                       llama.rope.freq_base f32              = 500000.000000
llama_model_loader: - kv  15:     llama.attention.layer_norm_rms_epsilon f32              = 0.000010
llama_model_loader: - kv  16:                 llama.attention.key_length u32              = 64
llama_model_loader: - kv  17:               llama.attention.value_length u32              = 64
llama_model_loader: - kv  18:                          general.file_type u32              = 15
llama_model_loader: - kv  19:                           llama.vocab_size u32              = 128256
llama_model_loader: - kv  20:                 llama.rope.dimension_count u32              = 64
llama_model_loader: - kv  21:                       tokenizer.ggml.model str              = gpt2
llama_model_loader: - kv  22:                         tokenizer.ggml.pre str              = llama-bpe
...

' }}
Using chat eos_token: <|eot_id|>
Using chat bos_token: <|begin_of_text|>
```

3.5 Performance Visualization

The system benchmarks performance across extraction, RAG queries, and translation tasks. RAG response time and output length are measured on sample queries (e.g., "What is Dr. X's contribution?"). Translation performance is measured by translating chunks to Arabic. Results are graphed in a 2x2 subplot grid:

4. PERFORMANCE ANALYSIS

4.1 Extraction Efficiency




```
Starting processing of files in folder: Dr.X
==================================================

Processing file: Dataset summaries and citations.docx
Successfully extracted text from Dataset summaries and citations.docx
Extraction time: 0.01 seconds
Number of chunks created: 14

Processing file: Loan amortisation schedule1.xlsx
Successfully extracted text from Loan amortisation schedule1.xlsx
Extraction time: 0.02 seconds
Number of chunks created: 13

Processing file: Loan analysis.xlsx
Successfully extracted text from Loan analysis.xlsx
Extraction time: 0.02 seconds
Number of chunks created: 37

Processing file: M.Sc. Applied Psychology.docx
Successfully extracted text from M.Sc. Applied Psychology.docx
Extraction time: 0.08 seconds
Number of chunks created: 287

Processing file: new-approaches-and-procedures-for-cancer-treatment.pdf
Successfully extracted text from new-approaches-and-procedures-for-cancer-treatment.pdf
Extraction time: 0.02 seconds
Number of chunks created: 127

Processing file: Ocean_ecogeochemistry_A_review.pdf
c:\mystuff\drx_project\venv\Lib\site-packages\openpyxl\worksheet\_reader.py:329: UserWarning:
  warn(msg)
Successfully extracted text from Ocean_ecogeochemistry_A_review.pdf
Extraction time: 0.11 seconds
Number of chunks created: 498

Processing file: party budget1.xlsx
Successfully extracted text from party budget1.xlsx
Extraction time: 0.03 seconds
Number of chunks created: 47

Processing file: Stats.docx
Successfully extracted text from Stats.docx
Extraction time: 0.01 seconds
Number of chunks created: 20

Processing file: The-Alchemist.pdf
Successfully extracted text from The-Alchemist.pdf
Extraction time: 0.08 seconds
Number of chunks created: 600

Processing file: The_Plan_of_the_Giza_Pyramids.pdf
Successfully extracted text from The_Plan_of_the_Giza_Pyramids.pdf
Extraction time: 0.03 seconds
Number of chunks created: 100

==================================================
All files processed!
Total number of chunks: 1743
Total processing time: 0.43 seconds
Extraction stats recorded for 10 files
```

The extraction job could handle different types of files, and its performance varied with file size and type. For example(above):

4.2 RAG Performance

The RAG system responded to sample queries with good efficiency, with response time of between 0.5 to 2 seconds, depending on query complexity and number of chunks returned. Token counts for responses were typically in the order of 50-150 tokens, indicating concise but informative output.

4.3 Translation Performance

Arabic translation achieved 100-200 tokens per second on average, where performance was limited by model inference speed and chunk length.

4.4 Summary Metrics

```
Batches: 100%|██████████| 1/1 [00:00<00:00, 97.10it/s]
Llama.generate: 512 prefix-match hit, remaining 510 prompt tokens to eval
llama_perf_context_print:        load time =    8374.03 ms
llama_perf_context_print: prompt eval time =    4666.60 ms /   510 tokens (    9.15 ms per token,   109.29 tokens per second)
llama_perf_context_print:        eval time =     533.81 ms /    21 runs   (   25.42 ms per token,    39.34 tokens per second)
llama_perf_context_print:       total time =    5239.16 ms /   531 tokens
⚡ Answer generated in 5.24 sec
Batches: 100%|██████████| 1/1 [00:00<00:00, 88.88it/s]
Llama.generate: 16 prefix-match hit, remaining 1037 prompt tokens to eval
llama_perf_context_print:        load time =    8374.03 ms
llama_perf_context_print: prompt eval time =    8996.33 ms /  1037 tokens (    8.68 ms per token,   115.27 tokens per second)
llama_perf_context_print:        eval time =    9885.56 ms /   395 runs   (   25.03 ms per token,    39.96 tokens per second)
llama_perf_context_print:       total time =   19819.52 ms /  1432 tokens
⚡ Answer generated in 19.83 sec
Batches: 100%|██████████| 1/1 [00:00<00:00, 97.14it/s]
Llama.generate: 16 prefix-match hit, remaining 1020 prompt tokens to eval
llama_perf_context_print:        load time =    8374.03 ms
llama_perf_context_print: prompt eval time =    8270.30 ms /  1020 tokens (    8.11 ms per token,   123.33 tokens per second)
llama_perf_context_print:        eval time =   12130.17 ms /   511 runs   (   23.74 ms per token,    42.13 tokens per second)
llama_perf_context_print:       total time =   21670.75 ms /  1531 tokens
Your input_length: 531 is bigger than 0.9 * max_length: 512. You might consider increasing your max_length manually, e.g. translator('...', max_length=400)
⚡ Answer generated in 21.67 sec
❌ Translation failed: index out of range in self
Batches: 100%|██████████| 1/1 [00:00<00:00, 39.36it/s]
Llama.generate: 512 prefix-match hit, remaining 523 prompt tokens to eval
llama_perf_context_print:        load time =    8374.03 ms
llama_perf_context_print: prompt eval time =    4561.06 ms /   523 tokens (    8.72 ms per token,   114.67 tokens per second)
llama_perf_context_print:        eval time =   12487.37 ms /   511 runs   (   24.44 ms per token,    40.92 tokens per second)
llama_perf_context_print:       total time =   18249.38 ms /  1034 tokens
Your input_length: 556 is bigger than 0.9 * max_length: 512. You might consider increasing your max_length manually, e.g. translator('...', max_length=400)
Your input_length: 546 is bigger than 0.9 * max_length: 512. You might consider increasing your max_length manually, e.g. translator('...', max_length=400)
Your input_length: 509 is bigger than 0.9 * max_length: 512. You might consider increasing your max_length manually, e.g. translator('...', max_length=400)
⚡ Answer generated in 18.25 sec
❌ Translation failed: index out of range in self
❌ Translation failed: index out of range in self
Your input_length: 529 is bigger than 0.9 * max_length: 512. You might consider increasing your max_length manually, e.g. translator('...', max_length=400)
Your input_length: 481 is bigger than 0.9 * max_length: 512. You might consider increasing your max_length manually, e.g. translator('...', max_length=400)
```

Summary token count was moderate, reflecting the system's ability to summarize effectively.

4.5 Qualitative Analysis

The system answered questions about Dr. X's contributions, research topics, datasets, and methods correctly. Quality, however, differed with the relevance of retrieved chunks and the reasoning capabilities of the LLaMA model.

5. Challenges and Limitations

5.1 Data Quality

Problem: Some files (e.g., Excel files with complex formatting) gave warnings during extraction, potentially excluding data.

Effect: Reduced completeness of the vector database.

Solution: Use aggressive preprocessing to handle formatting issues.

5.2 Computational Constraints

Problem: Local LLaMA model and FAISS indexing require high memory and CPU usage.

Effect: Slow performance on low-end devices.

Solution: Adjust chunk size or use cloud-based inference.

5.3 Translation Accuracy

Problem: Translation to Arabic occasionally introduced semantic errors.

Effect: Reduced reliability for multilingual applications.

Solution: Fine-tune translation model or use a specialized API.

5.4 Scalability

Issue: The flat FAISS index becomes inefficient for very large datasets.

Impact: High retrieval latency.

Solution: Try hierarchical or approximate nearest-neighbor indexing.

6. CONCLUSION AND FUTURE WORK

The RAG system effectively manages Dr. X's data, enabling efficient retrieval and context-sensitive response generation. Its key strong points include the modular design, support for multiple file formats, and extensive performance evaluation. However, data quality concerns, computational constraints, and translation accuracy reveal areas for enhancement.

Future directions include:

Enhancement of document preprocessing to handle complex formats.

Optimization of the system's scalability using sophisticated indexing techniques.

Adding a more powerful translation model to accommodate various languages.

Hosting the system on a cloud platform for greater accessibility.

This project demonstrates the potential of RAG systems for research and academic application, providing a foundation for further development.

References:

Hugging Face. (2023). Sentence Transformers Documentation.

Meta AI. (2023). LLaMA Model Specifications.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. EMNLP-IJCNLP.

FAISS Documentation. (2023). Meta AI Research.