The Enigmatic Research of Dr. X

### ****Section 1: Reading the Publications****

The first stage of the project involved collecting and organizing various publications provided in the “DR. X Files” folder. This folder contained multiple file types—PDF, DOCX, and CSV/XLSX—each requiring a different approach for content extraction. To efficiently handle these formats, three dedicated Jupyter notebooks were created: **Processing PDF file.ipynb**, **Processing DOCX file.ipynb**, and **Processing XLSX file.ipynb**. Each notebook is designed to read the files, extract the textual content, and prepare the documents for downstream tasks like text chunking and analysis.

### ****Section 2: Breaking Down the Publications****

Once the documents were read successfully, the next task was to break down the extracted text into manageable segments. The outputs from this process were saved in separate CSV files: **docx\_chunks\_text.csv**, **excel\_chunks\_text.csv**, and **pdf\_chunks\_text.csv**, which were later merged into a single Excel file named **ThreeFilesMerged.xlsx** for unified access and analysis. Each CSV file contains a column with the chunked text data.

**Section 3: Building a Vector Database**

This section focuses on transforming the processed text data into a searchable vector database. The merged file **ThreeFilesMerged.xlsx** served as the primary input. Using the Jupyter notebook **Nomic Embedding Model and RAG Q&A System.ipynb**, embeddings were generated and stored in a vector index for efficient semantic search and retrieval.

The notebook loads and prepares the data from the Excel file, generates embeddings using the **Nomic embedding model**, and stores them in a **FAISS** vector index. Metadata associated with each chunk (e.g., source, page number, and chunk number) was stored separately in a pickle file for easy reference and retrieval.

The main outputs of this step include:

* **vector\_index.faiss**: The FAISS-based vector database of chunk embeddings.
* **vector\_metadata.pkl**: A pickle file containing metadata linked to the index positions in the vector database.

**Section 4: Creating a RAG Q&A System**

In this final stage, the goal is to build a **Retrieval-Augmented Generation (RAG)** system that can answer user questions using the previously created vector database. The input remains the **ThreeFilesMerged.xlsx** file, and the process is implemented in the same notebook: **Nomic Embedding Model and RAG Q&A System.ipynb**.

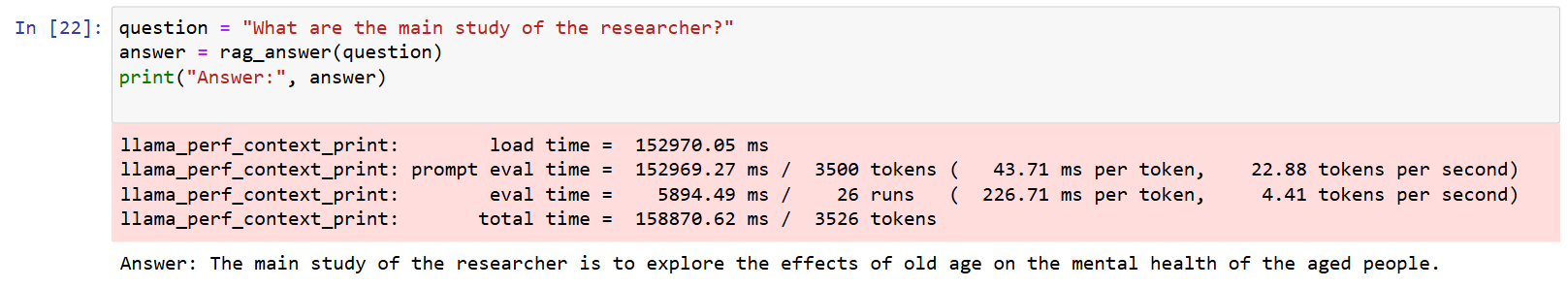
The process begins by loading the **FAISS vector index** and **associated metadata**. When a user submits a question, it is first embedded using the same Nomic model used during chunk embedding. The system then performs a similarity search on the vector database to retrieve the most relevant chunks. These chunks are passed to a **local LLaMA language model** to generate a coherent, context-aware answer.

The core steps are:

1. Load the vector index and metadata.
2. Embed the user's question using the Nomic model.
3. Retrieve the top-K most relevant chunks using vector similarity.
4. Load a local LLaMA model using llama-cpp-python for free, offline inference.
5. Generate a response based on the retrieved information—this is the core RAG operation.

To run the LLaMA model locally, download the model file from Hugging Face:

[https://huggingface.co/TheBloke/Llama-2-7B-Chat-GGUF/blob/main/**llama-2-7b-chat.Q4\_K\_M.gguf**](https://huggingface.co/TheBloke/Llama-2-7B-Chat-GGUF/blob/main/llama-2-7b-chat.Q4_K_M.gguf)

****Download the Model: **TheBloke/Llama-2-7B-Chat-GGUF**

### ****Section 5: Translating the Publications****

This section focuses on translating extracted content from the publications into Arabic. The translation process was implemented using the **Translation.ipynb** notebook, and applied to documents in **PDF**, **DOCX**, and **CSV** formats.

Two translation strategies were explored:

* **Chunk-based translation**: Initially translating entire chunks to preserve context.
* **Sentence-based translation**: Breaking content into sentences for improved fluency and translation accuracy.

#### **PDF Files: Worked**

**Input:** new-approaches-and-procedures-for-cancer-treatment.pdf

* **Chunk-based Output:** translated\_ar.txt
* **Sentence-based Output:** translated\_document\_full.txt

#### **DOCX File: Worked**

**Input:** Stats.docx

* **Sentence-based Output:** Stats\_translated.docx

#### **CSV File: Didn’t work ☹**

**Input:** excel\_chunks\_text.csv

* **Sentence-based Output (Selective Translation):** excel\_chunks\_text\_translated\_selective.csv

**Section 6: Finding the Main Ideas**

This section focuses on summarizing the content of various document formats to extract their core ideas. The process leverages both **abstractive** and **extractive** summarization models to ensure a balance between fluency and factual accuracy. Summarization was applied to DOCX, PDF, and CSV files using the following notebooks:

* **DOCX Input:** Stats.docx  
  Code: Summarization\_docx.ipynb
* **PDF Input:** new-approaches-and-procedures-for-cancer-treatment.pdf  
  Code: Summarization\_pdf.ipynb
* **CSV Input:** excel\_chunks\_text.csv  
  Code: Summarization\_Performance Measurement\_CSV.ipynb

To accommodate different summarization goals, multiple models were used:

* **PEGASUS**: Fluent and human-like summaries — ideal for narrative documents, overviews, and news-style content. Used as the **reference summary** in evaluation.
* **BART**: Balanced and factual summaries — great for accurate overviews.
* **T5**: Flexible, multi-purpose model — useful for customized summarization tasks.
* **TextRank & Luhn**: Extractive models — best for fact-focused summarization by selecting key sentences.
* **SpaCy (Custom)**: Fast, keyword-driven extraction — used for lightweight summarization.

**Note:**  
gensim was initially tested for extractive summarization but was not used due to version conflicts and unresolved dependency issues.

**Results of CSV Input:** excel\_chunks\_text.csv  
Code: Summarization\_Performance Measurement\_CSV.ipynb

To evaluate the performance of different summarization models, I used the PEGASUS-generated summaries as reference (since I don’t have human reference summary) and computed ROUGE scores (ROUGE-1, ROUGE-2, and ROUGE-L) for BART, T5, TextRank, Luhn, and SpaCy-based summaries as shown below:

ROUGE Scores PEGASUS with BART:

rouge1: Precision: 0.1860, Recall: 0.0471, F1: 0.0751

rouge2: Precision: 0.0000, Recall: 0.0000, F1: 0.0000

rougeL: Precision: 0.1860, Recall: 0.0471, F1: 0.0751

ROUGE Scores PEGASUS with T5:

rouge1: Precision: 0.0076, Recall: 0.0118, F1: 0.0093

rouge2: Precision: 0.0038, Recall: 0.0059, F1: 0.0047

rougeL: Precision: 0.0076, Recall: 0.0118, F1: 0.0093

ROUGE Scores PEGASUS with TextRank:

rouge1: Precision: 0.0089, Recall: 0.1647, F1: 0.0169

rouge2: Precision: 0.0006, Recall: 0.0118, F1: 0.0012

rougeL: Precision: 0.0057, Recall: 0.1059, F1: 0.0109

ROUGE Scores PEGASUS with Luhn:

rouge1: Precision: 0.0089, Recall: 0.1647, F1: 0.0169

rouge2: Precision: 0.0006, Recall: 0.0118, F1: 0.0012

rougeL: Precision: 0.0057, Recall: 0.1059, F1: 0.0109

ROUGE Scores PEGASUS with SpaCy (Custom):

rouge1: Precision: 0.0090, Recall: 0.1647, F1: 0.0170

rouge2: Precision: 0.0006, Recall: 0.0118, F1: 0.0012

rougeL: Precision: 0.0051, Recall: 0.0941, F1: 0.0097

Overall, the ROUGE score evaluation using PEGASUS as the reference highlights significant differences in summarization quality across the models. BART outperformed the others in terms of ROUGE-1 and ROUGE-L precision and F1-scores, indicating a better overlap with PEGASUS summaries at the unigram and longest common subsequence levels. However, its ROUGE-2 score was zero, suggesting that it failed to capture meaningful bigram-level coherence. T5 produced notably low scores across all metrics, reflecting limited content overlap with PEGASUS. The extractive models—TextRank, Luhn, and SpaCy (Custom)—showed very similar patterns, achieving higher recall than precision but overall low F1-scores. This suggests they extracted some relevant content but lacked fluency and completeness when compared to PEGASUS. Despite their low scores, these models may still offer value in quick, fact-based summarization where completeness is less critical. The findings reinforce PEGASUS’s strength in generating human-like, cohesive summaries and highlight the trade-offs between abstractive and extractive summarization approaches. **It is also important to note that if a human-written reference summary were used instead of PEGASUS, the evaluation results might differ, potentially favoring more factual or concise extractive summaries.**

**Section 7: Performance Measurement (Summarization)**

**Results of CSV Input:** excel\_chunks\_text.csv  
Code: Summarization\_Performance Measurement\_CSV.ipynb

In this section, I measured the performance of six summarization models—BART, T5, TextRank, Luhn, and SpaCy (Custom), and PEGASUS —using the input file excel\_chunks\_text.csv. The goal was to evaluate their efficiency in processing tokens and generating summaries.

The following metrics were recorded:

* Total tokens processed
* Time taken (in seconds)
* Processing speed (tokens per second)

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Tokens processed | Time taken | Tokens per second |
| BART | 71 | 7.1216 seconds | 9.97 |
| T5 | 2469 | 13.8581 seconds | 178.16 |
| TextRank | 2469 | 0.0593 seconds | 41660.75 |
| Luhn | 2469 | 0.0355 seconds | 69520.25 |
| SpaCy + Sentence Scoring | 2469 | 0.6239 seconds | 3957.07 |
| PEGASUS | 2469 | 28.46 seconds | 86.76 |

From a performance standpoint, extractive models like TextRank, Luhn, and SpaCy significantly outperformed abstractive models such as PEGASUS, T5, and BART in terms of processing speed. For example, Luhn handled over 69,000 tokens per second, making it the fastest, followed closely by TextRank. In contrast, PEGASUS, despite its high-quality, human-like output, processed tokens at a much slower rate (86.76 tokens/sec) due to its heavy computation.

This trade-off between quality and speed highlights an important consideration in summarization tasks: while abstractive models produce more fluent and nuanced summaries, extractive models offer tremendous efficiency, making them ideal for quick, large-scale processing tasks.