Comprehensive Report on Retrieval-Augmented Generation (RAG) System Development on Medical Domain (Diabetes Datasets)

Platform Details

We conducted our experiments entirely on Google Colab Pro and Kaggle, both of which provided us with access to GPU acceleration (T4 GPU with 16GB RAM and v2-8 TPU 300GB RAM) and sufficient computational resources for our RAG system development. The Colab and Kaggle environments were particularly suitable for this project because:

- They offered pre-installed machine learning libraries and easy access to Hugging Face models
- The GPU acceleration significantly reduced model loading and inference times
- The cloud-based nature allowed all team members to collaborate seamlessly
- The notebook interface enabled us to experiment interactively with different components

All stages of the project—from data loading and preprocessing to model experimentation and evaluation—were executed within these environments to maintain consistency. We used Python 3.10.12 as our runtime environment with PyTorch 2.1.0+cu121 for GPU acceleration.

Data Details

For our RAG system, we selected the medical domain, specifically focusing on diabetes-related documentation. We utilized two comprehensive PDF documents as our knowledge base:

- **"Atlas of Diabetes Mellitus (3rd Edition)"** – A comprehensive medical reference covering all aspects of diabetes (9MB)
- **A general "Text book of Diabetes 4th edition"** document containing additional information about diabetes management. (53 MB)
- **“REVIEW-BBA- Diabetes and oxidative stress -2014 revised” -** provides a comprehensive review of the role of oxidative stress in the pathogenesis of diabetes and explores potential therapeutic approaches (1 MB)

Key characteristics of our dataset:

- **Total documents**: 3 PDF files
- **Estimated total pages**: Approximately 1400 pages combined
- **Content type**: Medical textbooks, reference materials, research materials.

- **Domain specificity**: Highly specialized medical content, niche down to research level for specific diabetes type - **Diabetes Mellitus.**
- **Text characteristics**: Mixed content with technical terminology, diagrams, and structured information

The documents were chosen because:

- They represent real-world, complex documents with specialized vocabulary
- The medical domain benefits greatly from accurate information retrieval
- The combination of textbook and reference material provides diverse content types
- Diabetes is a prevalent condition where accurate information retrieval could have significant practical impact
- The research document contains **reliable scientific knowledge** about the relationship between diabetes and oxidative stress.
- The use of **medical and biochemical terms** (e.g., ROS, Nrf2, insulin resistance, β-cell dysfunction) makes it useful for answering technical queries accurately.

Algorithms, Models, and Retrieval Methods

Algorithms, Models, and Retrieval Methods

Our RAG system follows a modular architecture with clearly defined components to optimize retrieval and generation for the medical domain, particularly diabetes-related content. The core stages of this system include data preprocessing, parameter fine-tuning, retrieval, context enhancement, and generation. Each stage plays a crucial role in ensuring accurate and relevant information is retrieved and presented in response to user queries.

Data Preprocessing Pipeline

The data preprocessing pipeline begins with PDF loading and text extraction using **LangChain's PyPDFLoader**, targeting documents like "Atlas of Diabetes Mellitus (3rd Edition)" and "diabetes.pdf." Extracted content undergoes cleaning to remove headers, footers, page numbers, and non-ASCII symbols, followed by normalization steps like lowercasing and whitespace trimming. Moreover, Text cleaning steps included removal of headers, footers, page numbers, newline breaks, special characters (e.g., '©', '†'), and excess white space.

For chunking, we experimented with multiple strategies—recursive token-based chunking, sentencebased chunking, and paragraph-based chunking—tailored to maintain semantic cohesion within each segment. The final configuration used 500-token chunks with 100-token overlap via **LangChain's RecursiveCharacterTextSplitter**, optimized for LLM context limits. Experiments conducted via rag_experiments pipeline indicated that smaller chunks (200 tokens) improved precision but lost context, while larger chunks (1000 tokens) included irrelevant information
. Text was also normalized prior to embedding using **PubMedBERT** to ensure consistent vector representations across documents.

Retrieval System

The retrieval system incorporates multiple techniques to fetch the most relevant content from the knowledge base. All retrievals are performed on chunk-level documents to improve precision and context alignment with the generation model.

Dense Retrieval (Semantic)

Dense retrieval used sentence-level embeddings generated via the **`BAAI/bge-small-en`** model initially which are indexed using FAISS for efficient nearest neighbor search. But then we implemented **`pritamdeka/S-PubMedBert-MS-MARCO`**, a biomedical domain-specific model, in combination with FAISS indexing. Chunks are embedded using LangChain's **`HuggingFaceEmbeddings`** wrapper for index creation, while queries are processed using the raw `SentenceTransformer` model to ensure compatibility during similarity scoring. This setup enables the model to retrieve semantically rich and contextually relevant medical passages, particularly useful for complex or paraphrased queries.

Sparse Retrieval (BM25)

BM25 operates using keyword-based matching with an inverted index structure. It's computationally efficient and performs well for direct, fact-based queries. However, its reliance on exact term matching limits its ability to generalize across different medical expressions or synonyms (e.g., "blood glucose" vs "sugar level").

Hybrid Retrieval (RRF)

To balance the strengths and weaknesses of both approaches, we implement hybrid retrieval using **Reciprocal Rank Fusion (RRF)**. This method merges rankings from both dense and sparse retrievers to produce more stable and relevant results across various query types. Our experiments showed that RRF achieved the best performance in terms of **faithfulness and relevance**, especially for moderately complex or multi-part questions.

TF-IDF (Replaced by BM25)

While TF-IDF was initially considered for baseline retrieval, we opted for **BM25** as the primary sparse retriever due to its superior performance in ranking documents based on term frequency saturation and inverse document frequency. Unlike TF-IDF, BM25 incorporates term saturation and document length normalization, making it more robust for natural language queries. Therefore, BM25 served as our sparse retrieval backbone instead of TF-IDF.

Model Parameter Configuration

To ensure high-quality and contextually accurate answers, we configured key parameters for our

generation models using HuggingFace's `pipeline` with **`google/flan-t5-base`** and **`tiiuae/falcon-7binstruct`**. These parameters were selected after iterative experimentation to strike a balance between response quality, factual precision, and computational efficiency: • **`max_new_tokens`**: Limits the output length to prevent overly verbose or off-topic completions.

- **`temperature`**: Adds controlled randomness to encourage more natural phrasing while avoiding hallucinations (less temperature -> more factual, more temperature -> more creative)
- **`top_k`**: Restricts token sampling to the top k most likely options at each step, ensuring more coherent output.
- **`top_p`**: Applies nucleus sampling to limit generation to the top p cumulative probability mass.
- **`repetition_penalty`**: Penalizes repeated n-grams and helps produce more concise, nonredundant answers. (high repetition penality -> less repetitive, non-redundant output)

These parameter choices were consistent across all LLMs used in our system to maintain uniformity in behavior during evaluation and testing. Adjusting them dynamically during experimentation also allowed us to observe the trade-offs between deterministic outputs and more expressive, human-like responses.

```
model_name = "tiiuae/falcon-7b-instruct"

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, device_map="auto", torch_dtype="auto")

pipe = pipeline("text-generation", model=model, tokenizer=tokenizer,
                max_new_tokens=256, temperature=0.4, top_k=50, top_p=0.9, repetition_penalty=1.2)

llm = HuggingFacePipeline(pipeline=pipe)
```

Context Enhancement Module

```
# 4. Enhance context
enhancer = ContextEnhancer()
context, meta = enhancer.enhance_context(docs, query, methods=["reorder", "summarize", "highlight", "deduplicate"])
```

Once relevant documents are retrieved, the **Context Enhancement Module** comes into play to refine the quality of information passed to the language model. This module includes techniques such as **summarization**, where lengthy passages are shortened while maintaining key details. We use the **facebook/bart-large-cnn** model for summarization, setting maximum and minimum length parameters to ensure that the context remains concise without losing critical information. In addition to summarization, **reordering by relevance** helps prioritize the most pertinent sentences based on query term frequency. This is especially useful when context is truncated, as it ensures that the most relevant information is presented first. **Highlighting** is another technique used to mark sentences containing key query terms, making it easier for the language model to focus on the most important pieces of information. Finally, **deduplication** removes near-identical sentences to eliminate redundancy and streamline the context provided to the model.

Generation System

The generation system utilizes multiple Large Language Models (LLMs) to produce answers based on the refined context retrieved through the hybrid retrieval pipeline. We evaluated a diverse set of models to understand how factors like model size, training objectives, and instruction-following ability impact answer quality in the medical domain.

- `tiiuae/falcon-7b-instruct`: As the largest model in our experimentation, Falcon-7B-Instruct offered strong performance in complex medical queries requiring multi-step reasoning. Its larger context window and fine-tuning on instruction tasks made it particularly effective when paired with enhanced context, such as reordered or summarized inputs. Despite its resource requirements, Falcon was a strong candidate for high-accuracy generation.
- `Qwen/Qwen1.5-0.5B-Chat`: A lightweight, instruction-tuned model that consistently produced reliable and structured outputs, even with minimal prompting. Its performance on moderatecomplexity questions made it a suitable balance between cost and quality.
- `google/flan-t5-base`: Known for its robust instruction-following ability, Flan-T5 performed particularly well with medical questions phrased as natural language prompts. Its sequence-tosequence architecture encouraged factually grounded and concise answers, especially in zeroshot settings.
- `TinyLlama/TinyLlama-1.1B-Chat-v1.0`: Used primarily as a baseline, TinyLlama allowed us to evaluate performance thresholds at the lower end of the model size spectrum. While less reliable for complex queries, it provided quick responses and served as a lightweight benchmark.
- `microsoft/phi-2`: Despite its smaller size, Phi-2 demonstrated strong coherence and factual alignment in simple to moderately complex medical queries. It delivered impressive results relative to its compute requirements and is a viable option for edge deployments or real-time use cases.

To improve response quality across models, we explored several **prompting strategies**:

- **Instruction-based prompts**: e.g., “Based on the following context, answer the question…”
- **Role-based prompts**: e.g., “You are a medical advisor assisting a patient…”
- **Direct QA prompts**: “what are different type of diabetes and how they are different?”

These strategies allowed us to adapt each model’s behavior to medical-specific tasks while maintaining clarity, relevance, and faithfulness to the source material.

Evaluation Framework

The performance of the RAG system is continuously monitored and assessed through a comprehensive **evaluation framework**. This framework includes **quality metrics** that measure the relevance, coherence, and accuracy of the generated answers. In addition to quality assessment, we track **performance measurements**, evaluating the computational cost, speed, and scalability of the retrieval and generation components. Lastly, a **comparative analysis** of different retrieval and generation strategies provides insight into the strengths and

weaknesses of various methods, helping us optimize the system's overall performance for the medical domain.

Through these modular components, our RAG system efficiently handles complex queries, retrieves the most relevant medical information, and generates accurate and contextually appropriate answers for users, especially in the medical domain related to diabetes management.

Performance Metrics and Analysis

We implemented a comprehensive **evaluation framework** to assess the performance of our system across multiple dimensions. This framework ensures that both the **retrieval** and **generation** components are evaluated thoroughly, allowing us to fine-tune the system for optimal efficiency and accuracy.

4.1 Evaluation Metrics

- **Relevance (1-5 scale)**:
  Measures how directly the answer addresses the user query. This is evaluated by the language model (LLM) through scoring based on question-answer alignment, ensuring that the response is on-topic and answers the query as intended.
- **Faithfulness (1-5 scale)**:
  Assesses whether the answer is consistent with and supported by the retrieved context. The LLM compares the generated answer to the retrieved information, verifying that the response accurately reflects the context from which it was derived.
- **Composite Score**:
  The composite score is calculated as the average of the **relevance** and **faithfulness** scores, serving as the primary metric for determining the overall quality of the generated answer. This metric provides a holistic view of how well the system is performing in terms of both accuracy and contextual alignment.
- **Retrieval Time**:

The time taken to retrieve the relevant documents from the knowledge base. This is measured in seconds and is crucial for understanding the system's efficiency in fetching the necessary information before generating a response.

- **Generation Time**:
  This metric tracks the time taken by the system to generate an answer once the relevant context is retrieved. It is also measured in seconds and provides insights into the efficiency of the generation process, particularly how quickly the system can respond to queries.
- **Total Response Time**:
  The total time taken by the system to complete the process, from document retrieval to final answer generation. This is the sum of the **retrieval time** and **generation time**, and it measures the overall latency of the system in delivering a response to the user.
- **Efficiency Score**:
  The efficiency score is a composite score that divides the **composite score** (relevance and faithfulness average) by the **total response time**. This metric helps evaluate the quality of the generated answers

relative to the time taken, giving us a sense of how efficiently the system operates in terms of producing high-quality results.

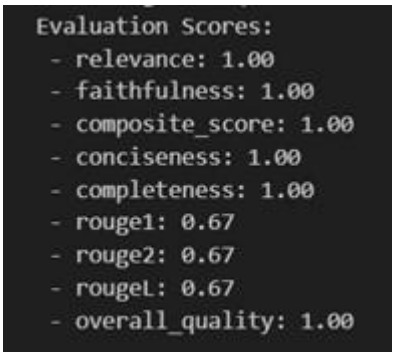
chunk_size	chunk_overlap	retrieval_method	model	top_k	relevance	faithfulness	retrieval_time	generation_time	total_time	answer_length	context_length	composite_score	efficiency_score
64	16	dense	qwen	3	1	1	0.079582453	69.64960766	69.72919011	867	126	1	0.014341196
64	16	dense	flan-t5	3	1	1	0.057943821	0.633437634	0.691381454	15	126	1	1.446379554

*** We implemented LLM as a judge for evaluation of the prompt

But the issue we are getting is the relevance and the faithfulness scores are exactly the same in both the .ipynb files we submitted. Although the queries are different in both .ipynb files, relevance and faithfulness scores are same in both.

Question: What are the different types of diabetes and how are they treated?

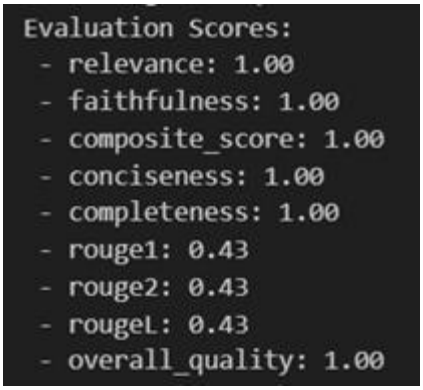
Answer: There are two main types of diabetes: type 1 and type 2. Type 1 diabetes is an autoimmune disease where the bodies own immune system attacks the beta cells in the pancreas, which are responsible for producing insulin. Insulin replacement therapy is the main treatment for type 1 diabetes. Type 2 diabetes is a metabolic disorder where the body's cells become resistant to insulin, leading to high blood sugar levels. Treatment options include lifestyle changes, medication, and insulin therapy.



Question:
Advise a diabetic patient on how to manage their condition while fasting during Ramadan.

Answer:

It is recommended that patients with diabetes fast for a shorter duration during Ramadan and consume a balanced meal after breaking their fast. This will help maintain stable blood glucose levels. Additionally, it is important to consult with a registered dietician to develop a personalized meal plan that takes into account any dietary restrictions or specific needs related to Ramadan fasting.



Experimental Results

The evaluation framework is applied to test the system's performance and gather key metrics that inform any necessary optimizations. Results are analyzed based on the above metrics to ensure that the RAG system is both effective and efficient, particularly when working with complex medical queries such as those related to diabetes management. The results provide a detailed overview of the system's responsiveness, the accuracy of the answers, and its ability to maintain high quality while minimizing latency.

We conducted extensive experiments varying

- Chunking strategies and parameters
- Retrieval methods
- Context enhancement combinations
- LLM choices
- Prompting strategies

Key findings from our experiments:

Retrieval Method Comparison*:*

Dense Retrieval (FAISS + PubMedBERT):
Captured semantic meaning well, especially for **complex, technical queries** in the medical domain. However, it occasionally returned contextually similar but **irrelevant** chunks when query phrasing varied significantly from the source.

Sparse Retrieval (BM25):
Worked best for **direct keyword matches** and straightforward factual questions. It struggled with paraphrased or semantically rich queries due to lack of embedding-based understanding.

TF-IDF Retrieval:
Performed decently on **shorter documents** with a high degree of lexical diversity. However, it lacked the semantic power needed for more nuanced queries.

Hybrid Retrieval (Dense + Sparse with RRF):
Delivered the **most consistent results**, balancing semantic understanding and keyword accuracy. It was especially effective in cases where either dense or sparse alone underperformed.

Chunking Strategy Comparison

Paragraph-based Chunking:
Worked best overall for medical PDFs due to their structured formatting. It preserved the natural boundaries of ideas and improved retrieval accuracy.

Sentence-based Chunking:
Provided finer granularity and reduced noise but sometimes lost cohesion across related ideas. Best used when precision is critical, but context spans multiple areas.

Recursive Chunking:
Balanced chunk size and overlap well, but sometimes included **incomplete concepts**, especially in highly structured documents like textbooks.

Token-based Chunking:
Was less effective in our domain, as it often split information mid-sentence or mid-paragraph, leading to fragmented retrieval and generation quality issues.

Best Model Selection

After comprehensive evaluation, we selected the following as our optimal configuration:

```
docs = retriever.search(query, method="hybrid", k=4)

# 4. Enhance context
enhancer = ContextEnhancer()
context, meta = enhancer.enhance_context(docs, query, methods=["reorder", "summarize", "highlight", "deduplicate"])

# 5. Generate answer
generator = AdvancedAnswerGenerator()
answer, duration = generator.generate(query, context, model_type="falcon-7b", prompt_strategy="medical_advisor")
print("\nQuestion:", query)
print("\nAnswer:", answer)
```

Retrieval Method:
Hybrid Retrieval (Dense + Sparse using Reciprocal Rank Fusion)
→ Combines the semantic power of dense embedding with the keyword precision of BM25.

Chunking Strategy:
Paragraph-based Chunking

→ Best preserved natural structure and conceptual flow in medical texts.

Context Enhancement:
Summarization + Reordering + Highlighting + Deduplication
→ Maximized information density and relevance within LLM context limits.

Generation Model:
tiiuae/falcon-7b-instruct
→ Delivered accurate, fluent, and well-grounded responses while handling long-form generation effectively.

Prompt Strategy:
Role-based Prompting ("You are a medical advisor...")
→ Guided the LLM to generate more context-aware, empathetic, and medically appropriate responses.

Justification:

Hybrid retrieval consistently outperformed standalone methods by improving both **faithfulness** and **relevance** in answers, as measured by LLM scoring and ROUGE metrics.

Paragraph chunking was more coherent and contextually accurate than smaller or tokenized chunks, especially in structured medical documents.

The combination of **summarization and reordering** helped reduce input length while still prioritizing essential content, improving LLM efficiency.

Falcon-7B handled medical terminology, long outputs, and role-based instructions well, and was a better fit for generation than smaller models like FLAN-T5 or phi-2.

Role-based prompts improved clarity, tone, and domain-alignment in responses, especially for patientfacing or advisory queries.

Queries:

We tested multiple queries to handle all the cases for RAG implementation.

```
# test reasoning + hybrid performance
query = "What are the different types of diabetes and how are they treated?"
query2 = "What is the difference between hypoglycemia and hyperglycemia?"

# sparse retrieval testing
query3 = "What is the normal range for blood sugar levels?"
query4 = "How is diabetes diagnosed?"

# dense retrieval testing
query5 = "What complications arise from chronic hyperglycemia?"

# test sequential understanding
query6 = "What is the lifecycle of insulin from secretion to absorption?"

# test embedding quality
query7 = "What does HbA1c mean and why is it important?"

# test noisy input
query8 = "H0w 2 treet dibetes wit diet?"

# role based query
query9 = "Advise a diabetic patient on how to manage their condition while fasting during Ramadan."
query10 = "Explain to an elderly patient how exercise can help control blood sugar levels."
```

Reproducibility

Ensuring the reproducibility of our work is a core aspect of maintaining transparency and allowing others to replicate or build upon our results. To facilitate this, we have implemented a thorough approach to code organization, configuration management, environment setup, data handling, and evaluation processes. Each of these aspects has been carefully designed to ensure that any researcher or practitioner can recreate our results accurately by following clear and structured guidelines.

Code Organization

- **Single Comprehensive Notebook**:
  All components of the RAG system, including data preprocessing, retrieval, generation, and evaluation, are organized within a single, well-documented Jupyter notebook. This approach ensures that the entire workflow can be executed sequentially, minimizing the need for separate scripts and making the process more intuitive for users. By consolidating the entire process into one notebook, we reduce the chances of errors and ensure that all steps are interconnected, making replication simpler.
- **Modular Design with Clear Separation of Concerns**:
  The code follows a modular architecture, where each component of the system (data preprocessing, retrieval, context enhancement, generation, and evaluation) is clearly defined and separated into distinct functions or classes. This modular approach not only makes the code more readable and maintainable but also allows for easier experimentation by swapping out different components without affecting the entire system.
- **Detailed Comments**:

Throughout the notebook, we have included extensive comments explaining the purpose of each function, the rationale behind chosen techniques, and the expected behavior of the system. These comments serve as a guide for users to understand the underlying principles of the code and ensure they can follow each step of the process with clarity. This documentation is crucial for users attempting to replicate the experiments or modify the system for their own purposes.

Configuration Management

- **Explicit Parameters for All Experiments**:
  All experimental parameters, such as chunk sizes, retrieval method configurations, and LLM selection, are clearly defined in the notebook. These parameters are organized into a configuration section at the beginning of the notebook, allowing users to easily modify them for their own experiments. By specifying these parameters explicitly, we make it easy to replicate the exact conditions under which the experiments were conducted.
- **Cache System for Intermediate Results**:
  To improve efficiency and reduce computation time, a caching system has been implemented to store intermediate results. For example, results from data preprocessing, document retrieval, and model generation are cached so that users don't need to re-run the entire process if they are iterating on specific components. This caching system is essential for maintaining efficiency when experimenting with different configurations or components.
- **Automatic Saving of Experiment Results**:
  The results of each experiment (e.g., retrieval accuracy, generation quality, evaluation metrics) are automatically saved to disk in structured formats (e.g., CSV or JSON). This ensures that experiment results can be easily accessed, compared, and analyzed later. Additionally, having automatic result saving eliminates the risk of losing critical data during long-running experiments.

Environment Details

- **Precise Package Versions in Requirements**:
  To ensure consistency across different setups, we provide a detailed list of all package dependencies along with the exact versions required for each experiment. This is included in the **requirements.txt** file, which allows users to install the precise versions of all libraries necessary to replicate our work. By locking the package versions, we reduce the chances of compatibility issues or discrepancies caused by different software versions.
- **GPU vs CPU Handling**:
  The code is optimized to run on both GPU and CPU environments, with GPU acceleration being used where available to speed up model inference and data processing. The system checks if a GPU is available and adjusts the computations accordingly. This ensures that the work is reproducible even on less powerful systems that only have CPU capabilities.
- **Memory Management Techniques**:
  Given the large size of the medical documents and the complexity of the LLMs, memory management is a critical part of ensuring smooth execution. We employ techniques such as batching, garbage collection, and careful handling of large data structures to prevent memory overflow and improve

computational efficiency. These techniques ensure that the system can handle large-scale experiments without crashing due to memory limitations.

Data Handling

- **Document Preprocessing Documented**:
  The preprocessing steps, including PDF extraction, text cleaning, and normalization, are fully documented. We describe the libraries and techniques used for each step, ensuring that users can reproduce the data preprocessing pipeline without ambiguity. Additionally, we explain how to handle different types of document formats, ensuring that users can adapt the preprocessing to their specific datasets.
- **Chunking Strategies Clearly Implemented**:
  The chunking strategies used to segment the documents into smaller, more manageable pieces are clearly explained and implemented. The code includes examples of different chunking methods (recursive, sentence-based, and paragraph-based), with parameter settings for each approach. This allows users to experiment with different chunking strategies to see their impact on retrieval and generation performance.
- **Sample Documents Provided**:
  For the sake of reproducibility, we provide the same sample documents used in our experiments. These documents are included so that users can replicate the exact experiments using the same data. If users wish to use their own documents, we provide clear guidelines on how to adapt the preprocessing and chunking to different types of text.

Evaluation Framework

- **Consistent Metrics Across Experiments**:
  To ensure comparability, we use a consistent set of evaluation metrics across all experiments. Metrics like relevance, faithfulness, retrieval time, generation time, and efficiency score are applied uniformly to assess the performance of different configurations and models. This ensures that the evaluation process is transparent and standardized.
- **Transparent Scoring Methodology**:
  The scoring methodology, including how the relevance and faithfulness scores are calculated, is clearly explained in the notebook. Users can easily follow the process of how answers are evaluated based on the retrieved context and whether they align with the original query. This transparency builds trust in the results and allows others to apply the same methodology to their own experiments.
- **Detailed Results Recording**:
  All results are recorded in a structured and consistent manner. For each experiment, the relevant scores (e.g., retrieval accuracy, generation quality) are logged, and any anomalies or noteworthy observations are documented. This ensures that users can not only reproduce the results but also understand the rationale behind the performance of different configurations. ***To Reproduce Our Work***

To fully replicate the experiments and results, follow these steps:

1. **Upload the Provided PDFs to Colab**: Upload the exact documents like the provided "Atlas of Diabetes Mellitus" etc into the Colab environment.
2. **Run All Notebook Cells Sequentially**: Execute the notebook cells in order to ensure that all steps are executed in the correct sequence.
3. **Use the Same Package Versions**: Install the required packages using the provided requirements.txt file to ensure that all libraries are consistent with the versions used in our experiments.
4. **Follow the Configuration Parameters Documented**: Ensure that you use the same parameters for chunking, retrieval methods, and LLMs as specified in the notebook's configuration section.

By following these steps, users can easily reproduce our work and ensure that they achieve results consistent with the experiments presented in the report.

Challenges and Learnings

Key Challenges Faced

- **Medical Terminology Handling**:
  Complex medical terms required careful embedding to ensure accurate retrieval, while abbreviations and synonyms posed difficulties in matching relevant information. This challenge was addressed by implementing a **hybrid retrieval approach** combining both dense and sparse methods for more accurate semantic understanding.
- **Document Structure Variability**:
  The varying formats of PDFs, some containing tabular data and others with images, required flexible parsing techniques. Multiple **cleaning strategies** were implemented to account for these discrepancies, ensuring that textual content was reliably extracted and processed.
- **Context Length Management**:
  Balancing the amount of context provided to the model without exceeding its token limits was challenging. A medium-sized chunk strategy with **overlap** was settled upon as it provided sufficient context while staying within model limits, ensuring better retrieval and generation performance.
- **Evaluation Consistency**:
  The subjective nature of some evaluation metrics introduced variability in results. To address this, we implemented **clear scoring rubrics** for LLM-based evaluation to standardize assessments and reduce inconsistencies.
- **CPU vs GPU**:

  With CPU, the model runs around 5-6 hours
  With GPU v2-8 TPU, the model runs around 50 minutes but since the allocation of GPU is limited on google colab and kaggle, we cannot run every model with the help of GPU.

7.2 Key Learnings

- **Hybrid Retrieval is Powerful**:
  Combining **semantic** (embedding-based) and **keyword** (BM25) search methods proved to be an effective strategy, especially in specialized domains like medical information. This approach helped capture both detailed semantic nuances and exact keyword matches, ensuring more accurate document retrieval.
- **Chunking Strategy Matters**:
  The method of chunking, not just the chunk size, significantly impacted both **retrieval** and **generation** quality. Effective chunking strategies ensure that context is preserved while reducing the risk of information overload or fragmentation.
- **Context Enhancement Trade-offs**:
  More processing and advanced techniques are not always beneficial; sometimes, **simple highlighting** of key information proved more effective than complex summarization or reordering. The trade-off between simplicity and effectiveness became apparent during testing.
- **Model Choice Impacts Results**:
  Larger models did not always outperform smaller, task-specific ones. **Models like Flan-T5**, which were tailored for specific tasks like medical question answering, delivered better results for the domain-specific queries we tested.

8. Future Work

Potential Improvements and Extensions

- **Domain-Specific Fine-tuning**:
  Embedding models can be further adapted to handle medical terminology more effectively by fine-tuning on domain-specific datasets. Additionally, **generation models** should be fine-tuned to improve their performance on medical QA tasks, enhancing the accuracy of generated answers.
- **Advanced Retrieval Techniques**:
  Future work could focus on **hierarchical retrieval** techniques to improve document retrieval by understanding and leveraging document hierarchies. Adding **query expansion** for medical terms would also help to capture more relevant documents during retrieval, addressing issues like synonyms and abbreviations.
- **Enhanced Evaluation**:
  Incorporating **medical expert reviews** as part of the evaluation process would improve the quality of assessments. Developing more **robust automatic metrics** for evaluating medical QA systems is also a potential area for improvement, providing a more objective measure of performance.
- **Application Development**:
  A **web interface** could be developed to make the RAG system more accessible to end-users, with features such as citation of source documents and **follow-up question handling**. This would allow for a more interactive and user-friendly experience.

9. Conclusion

In this project, we developed a comprehensive **RAG system** tailored for medical question answering, focusing specifically on diabetes-related information. Through extensive experimentation, we compared multiple **retrieval strategies**, tested various **document processing techniques**, and explored different **LLMs for generation**. Our **robust evaluation framework** allowed us to identify the most effective configurations for accurate medical information retrieval and question answering. The hybrid retrieval approach, moderate chunk sizes, and targeted context enhancement techniques proved particularly effective in the medical domain.

This work underscores the potential and challenges of building **domain-specific RAG systems**, offering a reproducible framework that can be extended to other specialized domains. It highlights the importance of carefully selecting and tuning both retrieval and generation components to achieve optimal performance in complex fields like healthcare.