# Optimized Retrieval-Augmented Generation (RAG) Pipeline Implementation

The goal of this project is to implement an optimized Retrieval-Augmented Generation (RAG) pipeline.

This pipeline enables users to interact with semi-structured data in multiple PDF files by extracting, chunking, embedding, and storing the data for efficient retrieval. The system answers user queries, performs comparisons, and generates accurate responses using an LLM (Language Learning Model).

Key aspects include data ingestion, query handling, embedding optimization, and response generation.

This document outlines the core components and optimizations incorporated in the system for improved

performance and scalability.

## 1. Data Ingestion & Embedding

- Preprocessing: Extract text from PDFs and filter irrelevant content (headers, footers).
- Dynamic Chunking: Text is segmented into logical chunks to improve data granularity and retrieval performance.
- Embedding: Convert text chunks into vector embeddings using pre-trained models.
- Metadata Storage: Attach relevant metadata such as page numbers for better context in query responses.

## 2. Query Handling & Optimization

- Hybrid Search: Combines keyword-based search with vector similarity search for more relevant

results.

- Query Embedding: Converts user queries into vector form to retrieve the most relevant chunks of data.

- Optimized Retrieval: Retrieval methods are tuned with k-value adjustments, multi-pass retrieval, and cache for frequent queries.

- Response Generation: A conversational agent integrates the retrieved data to generate detailed, contextually accurate responses.

## 3. Comparison Queries

- Comparison Handling: Extracts key data fields such as unemployment rates, degree types, and compares across different PDF sources.

- Table Extraction: Utilizes libraries like Camelot to parse tables from PDFs.

- Aggregation: Data from multiple PDFs is aggregated, compared, and displayed in a structured format such as tables or bullet points.

## 4. LLM Integration & Optimizations

- Contextual Prompts: Injects relevant chunks into prompts to guide the model in producing accurate, data-backed responses.

- Model Selection: Uses models like GPT-4 for accuracy and GPT-3.5 for cost-effective use cases.

- Output Control: Ensures responses are concise and factually grounded by incorporating retrieved data directly.

## 5. Scalability and Performance Optimization

- Pinecone Integration: For large-scale vector database management, switching to Pinecone ensures fast retrieval and efficient storage.

- Asynchronous Requests: Asynchronous handling of requests reduces blocking time and improves user experience.

- Distributed Architecture: Scalable deployment via Kubernetes for cloud-based systems, ensuring reliability and high availability.

- Caching: Frequently used queries are cached to reduce repeated computation and improve system performance.

## 6. Deployment and Monitoring

- Docker Deployment: The backend is containerized using Docker to ensure smooth deployment across different environments.

- Streamlit Frontend: The frontend uses Streamlit for quick user interaction, with cloud-based deployment via Streamlit Cloud.

- Monitoring: Integrated logging and monitoring tools track the performance of queries, system errors, and user interactions.