Multimodal Retrieval-Augmented Generation (RAG) and VQA

A PROJECT REPORT

Submitted by

Roll Number	Registration Number	Student Code	Student Name
22010301155	22013002545	BWU/BCA/22/166	Shatadru Sarkar
22010301167	22013002557	BWU/BCA/22/178	Rupsa Saha
22010301135	22013002525	BWU/BCA/22/146	Shrabani Halder
22010301149	22013002539	BWU/BCA/22/160	Anwesha Samanta
22010301144	22013002534	BWU/BCA/22/155	Salma Sultana
22010301119	22013002505	BWU/BCA/22/126	Paramita Das

in partial fulfillment for the award of the degree

Of

BACHELOR OF COMPUTER APPLICATION

IN

Department of COMPUTATIONAL SCIENCE

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata - 700 125

JULY & 2025

Multimodal Retrieval-Augmented Generation (RAG) and VQA

Submitted by

Roll Number	Registration Number	Student Code	Student Name
22010301155	22013002545	BWU/BCA/22/166	Shatadru Sarkar
22010301167	22013002557	BWU/BCA/22/178	Rupsa Saha
22010301135	22013002525	BWU/BCA/22/146	Shrabani Halder
22010301149	22013002539	BWU/BCA/22/160	Anwesha Samanta
22010301144	22013002534	BWU/BCA/22/155	Salma Sultana
22010301119	22013002505	BWU/BCA/22/126	Paramita Das

in partial fulfillment for the award of the degree

Of

BACHELOR OF COMPUTER APPLICATION

IN

Department of COMPUTATIONAL SCIENCE

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata - 700 125

BRAINWARE UNIVERSITY

398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata - 700 125

Department of COMPUTATIONAL SCIENCE

BONAFIDE CERTIFICATE

Certified that this project report **Multimodal Retrieval-Augmented Generation (RAG) and VQA** is the bonafide work of "**Shatadru Sarkar** (Roll No:22010301155)**, Rupsa Saha** (Roll No:22010301167)**, Shrabani Halder** (Roll No:22010301135)**, Anwesha Samanta** (Roll No:22010301149)**, Salma Sultana** (Roll No:22010301144)**, Paramita Das** (Roll No:22010301119)" who carried out the project work under my supervision.

SIGNATURE
DR. JAYANTA AICH
HEAD OF THE DEPARTMENT
COMPUTATIONAL SCIENCE
398, Ramkrishnapur Road, Barasat,
North 24 Parganas, Kolkata - 700 125

SIGNATURE
SOURAV SADHUKHAN
SUPERVISOR
ASSISTANT PROFESSOR
COMPUTATIONAL SCIENCE

ACKNOWLEDGEMENT

Project Title: Multimodal Retrieval-Augmented Generation (RAG) and VQA

Project Group ID: BCA22C005

We, the undersigned project members, would like to express our heartfelt gratitude to [Mr. Sourav Sadhukhan, Assistant Professor], Department of Computational Sciences, Brainware University, for their invaluable guidance, support, and motivation throughout the course of this project. Their expert advice and insightful suggestions were instrumental in the successful completion of our work.

We are also immensely thankful to Dr. Jayanta Aich, Head of the Department of Computational Sciences, for providing the required resources and encouragement necessary for this project.

We acknowledge the cooperation and support of all faculty members and staff of the Department of Computational Sciences. Their input helped us remain on track and improve the quality of our research.

We extend our special thanks to our families, friends, and peers for their continuous encouragement and support during this academic journey and for motivating us to achieve our goals.

Lastly, we express our sincere gratitude to Brainware University for giving us the opportunity to undertake this project as a part of our academic curriculum.

Project Members:

1. BWU/BCA/22/166- Shatadru Sarkar
2. BWU/BCA/22/178- Rupsa Saha
3. BWU/BCA/22/146- Shrabani Halder
4. BWU/BCA/22/160- Anwesha Samanta
5. BWU/BCA/22/155- Salma Sultana
6. BWU/BCA/22/126- Paramita Das

Date:19.05.2025
Department of Computational Sciences

Brainware University

ABSTRACT

Multimodal Retrieval-Augmented Generation (RAG) and VQA refers to an approach that combines retrieval-based methods with generative models, but extends beyond text to handle multiple data types (like images, audio, or video). In this framework, the system first retrieves relevant information from external sources (e.g., documents, databases) and then generates responses based on both the retrieved content and the input query. The "multimodal" aspect means that the model can process and integrate information from various modalities (text, images, etc.) to provide more accurate, context-aware outputs.

This approach is useful in applications like answering questions that involve both text and images, or generating captions for visual content based on a combination of text and visual retrievals.

Table of Contents

List of Figures

List of Diagram

Chapter-1: Introduction

Multimodal Retrieval-Augmented Generation (RAG): Introduction:

Multimodal Retrieval-Augmented Generation (RAG) is an advanced framework in the field of natural language processing (NLP) and machine learning that integrates information retrieval techniques with generative models to handle multimodal data. It combines text, images, videos, and potentially other forms of data, allowing the model to search for relevant information across different modalities to enhance the generation process.

Key Concepts

1. **Multimodal Data**: Multimodal systems process and generate outputs based on data that involves multiple modes (e.g., text, images, audio, video). This contrasts with traditional approaches that rely on a single data modality.
2. **Retrieval-Augmented Generation (RAG)**: In standard RAG, a model retrieves relevant information from a large knowledge base or document store to enrich the generative process. The generative model, typically a Transformer-based architecture, uses this retrieved information to produce more informed and contextually relevant outputs.
3. **Multimodal RAG**: This concept extends the original RAG framework by incorporating multimodal inputs. Instead of just querying text-based documents, a multimodal RAG model retrieves information from a variety of sources — such as images, videos, or other sensory data — to improve the generation process. This results in a more contextually aware and accurate generation, especially in tasks that require understanding of both visual and textual information.

How Multimodal RAG Works:

Input Processing:

- The input can include various types of data (e.g., a combination of text and images). For example, the input might consist of a question that references both a concept (e.g., a famous painting) and a visual input (e.g., an image of that painting).

Multimodal Retrieval:

- The model retrieves relevant pieces of information from a multimodal database. This can involve querying:
    - Textual data (e.g., documents, articles, or FAQs).
    - Visual data (e.g., images or videos that might provide context or related content).
    - Audio or other forms of data (less common but possible in certain applications).

Fusion of Modalities:

- The retrieved data from different modalities is combined into a unified representation. This step typically involves aligning or embedding different types of data (such as transforming images into text-based features via CNNs or ViTs, or translating audio into text using speech recognition models).

Generation:

- A generative model (like GPT, BERT, or a variant) uses this multimodal, enriched representation to generate a relevant output. This might involve generating answers to questions, summarizing content, or creating captions for images.

Applications of Multimodal RAG:

1. **Visual Question Answering (VQA)**: In VQA, a model can answer questions about images. Multimodal RAG would allow the model to first retrieve relevant visual context from a database and then use that information to generate an accurate, contextual answer.
2. **Image Captioning with Context**: By retrieving relevant textual or visual data, a multimodal RAG model can generate more contextually enriched captions for images or videos.
3. **Medical Imaging**: In medical applications, a multimodal RAG system could retrieve relevant clinical documents, studies, or medical images (e.g., X-rays, MRIs) to assist in generating accurate diagnostic descriptions or recommendations.
4. **Robotics and Autonomous Systems**: Robots can leverage multimodal RAG to interpret visual and auditory inputs, retrieve relevant contextual information (such as manuals, diagrams, or sound signals), and generate natural language commands or responses.
5. **Content Creation and Editing**: Multimodal RAG can assist in generating multimedia content, such as automatically generating a video summary with both visual highlights and a narrative voiceover, by integrating visual and textual data from multiple sources.

Challenges and Considerations:

1. **Alignment of Modalities**: One of the significant challenges in multimodal systems is effectively aligning and integrating information from different modalities. For instance, images and text are inherently different, and ensuring they contribute to the same contextual understanding can be complex.
2. **Data Availability**: Multimodal RAG models often require large, diverse datasets that contain both labeled textual and visual information. Building such datasets is resource-intensive.
3. **Scalability**: As the amount of data grows, retrieving the most relevant information in a computationally efficient manner across modalities becomes a critical challenge.
4. **Bias and Fairness**: Like other AI systems, multimodal RAG models are susceptible to biases in the training data. Ensuring fairness and avoiding harmful outputs in multimodal contexts is crucial.

Chapter-2: Objective

The objective of Multimodal Retrieval-Augmented Generation (RAG) is to enhance the performance of generative models by integrating and utilizing information from multiple modalities (such as text, images, audio, video, etc.) to generate more accurate, informative, and contextually relevant outputs. The goal is to leverage multimodal information retrieval to improve the quality and relevance of generated content, particularly when tasks require understanding or referencing data from different sources and formats.

Key Objectives of Multimodal RAG:

1. **Enhancing Contextual Understanding**:
    - **Objective**: To deepen the model's understanding of the input by combining information from multiple modalities (e.g., images, text, video). This helps the model generate more precise and contextually aware responses.
    - **Example**: For a visual question answering (VQA) task, the model should not only rely on textual knowledge but also incorporate visual context to provide better answers.

2. **Improving Information Retrieval**:

- **Objective**: To augment generation by retrieving relevant data not only from text but also from other modalities. By retrieving visual data, audio clips, or even videos, the model can ground its generation in richer, multimodal content.
- **Example**: In a captioning task, a model might retrieve both related images and textual descriptions to generate a more accurate and informative caption.

3. **Context-Aware Generation**:

- **Objective**: To ensure that the generated content is more specific, relevant, and aligned with the context of the input. By pulling in multimodal data from the retrieval step, the generation process becomes more tailored and precise.
- **Example**: When generating a response to a question about a movie, the model could retrieve both textual information (e.g., plot summaries) and visual data (e.g., images or trailers) to provide a more well-rounded answer.

4. **Improving Task Performance in Multimodal Settings**:
    - **Objective**: To enhance performance on tasks that inherently require multimodal data, such as image captioning, video summarization, and multimodal dialogue systems. By integrating different modalities into the generation process, RAG models can handle complex multimodal tasks more effectively.
    - **Example**: In a multimodal dialogue system, the system may need to interpret both spoken language and visual inputs (such as gestures or facial expressions) to provide a more accurate and coherent response.

5. **Bridging the Gap Between Modalities**:

- **Objective**: To create a system that can effectively bridge the gap between different data types (e.g., translating visual content into textual descriptions, or vice versa) to facilitate richer interactions and more intelligent responses.
- **Example**: A model might take a combination of textual input (such as a question) and an image (e.g., of a landmark), retrieve relevant documents and images, and generate a detailed, informative answer combining both the visual and textual data.

6. **Scalability in Multimodal Retrieval**:

- **Objective**: To retrieve relevant multimodal information at scale. As data grows across different modalities, ensuring efficient and scalable retrieval from large multimodal databases becomes essential.
- **Example**: A large-scale multimedia search engine that retrieves relevant images, videos, and articles in response to a query could benefit from a multimodal RAG approach to provide more comprehensive search results.
- **Supporting Complex Real-World Applications:**

    - **Objective**: To support complex real-world applications that involve multimodal data, such as healthcare, autonomous systems, creative industries, and robotics.
    - **Example**: In healthcare, a model might retrieve relevant medical images (e.g., X-rays) and textual records (e.g., patient history) to generate a diagnostic recommendation.

Overall Goal of Multimodal RAG:

The overarching goal of **Multimodal RAG** is to improve the quality and reliability of generative outputs by effectively incorporating diverse modalities into the information retrieval and generation process. This leads to systems that are **more adaptable, intelligent, and context-aware**, with the ability to handle richer, more complex tasks involving multiple types of data sources, thus enabling more sophisticated real-world applications.

Chapter-3: Planning

- **<u>Planning of Multimodal RAG (Retrieval-Augmented Generation)</u>**

- Planning a Multimodal RAG system involves multiple stages and components to ensure that the retrieval and generation processes work seamlessly with multiple types of data (e.g., text, images, audio, video). The planning process focuses on organizing how the system will handle multimodal inputs, efficiently retrieve relevant data, and generate coherent and contextually accurate outputs.

Below is a structured plan for building and deploying a Multimodal RAG system:

- **Define System Objectives and Scope**

    - **Objective**: Establish the main goal of the multimodal RAG system, including which tasks it will address (e.g., Visual Question Answering, Image Captioning, Multimodal Dialogue Systems).
    - **Example**: If the system is for medical diagnostics, the objective may be to combine textual patient records with medical images (e.g., X-rays) to provide diagnostic recommendations.

Questions to Consider:

- What specific modalities will the system handle (e.g., text, images, video, audio)?
- What is the primary use case (e.g., question answering, summarization, creative content generation)?
- Who are the end users (e.g., medical professionals, general consumers, researchers)?

- **Data Collection and Preprocessing**

    - **Objective**: Gather, clean, and preprocess multimodal data for both retrieval and generation tasks.
    - **Data Types**: Collect and annotate data for each modality that the system will process (text, images, videos, etc.).

Key Steps:

- **Text Data**: Gather documents, FAQs, technical papers, or other relevant text corpora. Text should be processed into a structured format (e.g., tokenization, embeddings).
- **Image and Video Data**: Collect a diverse set of images and videos with annotations (e.g., object detection, captions). Preprocess these into feature vectors using models like CNNs (e.g., ResNet, EfficientNet) or Vision Transformers (ViTs).
- **Audio Data**: If applicable, collect audio or speech data and convert it into a textual form (e.g., via automatic speech recognition).

Example: For a medical RAG system, the data might include annotated medical records (text), diagnostic images (e.g., X-rays), and clinical reports.

- **Design the Multimodal Retrieval Mechanism**

    - **Objective**: Implement an effective retrieval strategy to fetch relevant data from multiple modalities (text, images, video, etc.) based on a query.

Key Components:

- **Multimodal Indexing**: Create indexes for each modality. For text, this could be a traditional document retrieval index (e.g., using Elasticsearch). For images and video, create feature embeddings using pre-trained CNNs or ViTs.
- **Query Processing**: When a query is received (e.g., a question or input image), decide how to process it:
    - If the query is textual, retrieve relevant text documents, then incorporate multimodal context if needed.
    - If the query is multimodal (e.g., a question about a specific image), retrieve both relevant text and visual content.

  Fusion of Modalities: For effective retrieval, you may need to combine embeddings from multiple modalities into a unified representation (e.g., a joint embedding space using techniques like cross-modal transformers or late fusion techniques).

Example: For a "Visual Question Answering" (VQA) task, the retrieval system should first retrieve relevant text (e.g., related documents or descriptions) and images (e.g., similar photos) and then use this information to augment the generation phase.

- **Design the Multimodal Fusion Model**

    - ***Objective**:* Design how the different modalities (text, image, video, audio) will be fused together for the generation task

Fusion Techniques:

- **Early Fusion**: Combine raw data from different modalities early in the model pipeline, often through joint embeddings. For example, using a Transformer-based architecture that processes text and visual inputs simultaneously.
- **Late Fusion**: Process each modality separately and combine their outputs at a later stage, such as at the decision or generation stage. This could involve combining different modality-specific representations (e.g., image embeddings and text embeddings) before generating a final output.

- **Multimodal Transformers**: Models like CLIP (Contrastive Language–Image Pretraining) or DALL·E use transformers to jointly process textual and visual information by aligning the text and image representations into a shared space.

Example: For a system that generates captions from both image and text data, the model would combine the visual and textual features to output a more descriptive caption.

2. Develop the Generation Model

- **Objective**: Build the generative model that will output relevant text (answers, summaries, descriptions) based on the retrieved multimodal data.

Steps to Consider:

- **Text Generation**: Use a language model (e.g., GPT, T5) to generate text based on the retrieved multimodal contex**Visual-to-Text Generation**: For tasks like image captioning or VQA, leverage models that can generate descriptive text from images (e.g., Image-to-Text models or
  Vision-Language models like BLIP or Flamingo).
- **Multimodal Training**: Train the model to handle and generate outputs based on multimodal inputs. This may involve pretraining on large-scale multimodal datasets (e.g., COCO, Visual Genome) and fine-tuning on task-specific datasets.

Example: For a dialogue system, the model should generate contextually relevant responses by integrating retrieved information (e.g., a user's question, images, previous conversation history) and providing a natural, informed response.

3. Model Evaluation

- **Objective**: Evaluate the performance of the multimodal RAG system on various metrics to ensure that it can handle multimodal tasks effectively.

Key Evaluation Metrics:

- **Accuracy**: For tasks like VQA or image captioning, evaluate how well the generated outputs align with ground truth data (e.g., accuracy of answers, BLEU score for captions).
- **Coherence**: Assess whether the generated text is coherent and relevant to all retrieved modalities.
- **Multimodal Consistency**: Ensure that the system can accurately combine information across modalities. For example, a generated caption should match the content of the image, and answers to questions should be grounded in both textual and visual data.
- **User Experience**: In applications like dialogue systems, assess how well the system understands multimodal cues and generates human-like, contextually appropriate responses.

4. **Deployment and Maintenance**

- **Objective**: Deploy the multimodal RAG system into a real-world environment, ensuring it performs well on live data.

Key Considerations:

- **Scalability**: Ensure the retrieval system can scale efficiently as the multimodal data grows. This may involve using cloud-based systems or distributed retrieval architectures.
- **Latency**: Optimize the system to provide fast response times, especially when integrating multiple modalities in real-time systems (e.g., a live multimodal customer service bot).
- **Continuous Improvement**: Regularly update the retrieval and generation models with new data to improve accuracy and adapt to changing user requirements. Consider adding feedback loops where users can correct generated outputs.

- **User Interface and Interaction**

- **Objective**: Design user interfaces (UIs) that can handle multimodal input and display multimodal output in a user-friendly manner.

Components:

- **Input Interfaces**: Support for users to input multiple types of data, such as text, images, or voice. This could include a chat interface with support for uploading images or speaking.
- **Output Interfaces**: Ensure that the generated content is presented in an easily understandable format, such as showing images alongside text or generating video clips based on textual descriptions.

Example: In a multimodal VQA system, users can upload an image and ask a question, and the system responds with a textual answer and may also highlight parts of the image that correspond to the answer.

Chapter-4: Requirement Analysis

Requirement Analysis of Multimodal Retrieval-Augmented Generation (RAG)

The requirement analysis phase for building a Multimodal Retrieval-Augmented Generation (RAG) system is crucial for understanding the functional and non-functional needs of the system, as well as identifying the necessary resources, technologies, and constraints. A clear and detailed analysis ensures that the system can effectively handle multimodal data and generate relevant and accurate responses by retrieving the right information from various sources. This phase involves analyzing the user needs, technical requirements, data requirements, and system constraints.

1. Functional Requirements

These define the primary functions that the Multimodal RAG system should perform, focusing on how the system will interact with users and handle multimodal data.

Key Functional Requirements:

1. Multimodal Data Retrieval:
    - The system should retrieve relevant data across different modalities (text, images, audio, etc.). It may need to search databases, documents, or multimedia content sources to pull relevant information for the task at hand.
    - Text-based Retrieval: Query relevant text documents, FAQs, articles, or research papers.
    - Image/Video Retrieval: Retrieve images or videos that are related to the query, either based on content or metadata (e.g., object recognition or visual features).
    - Cross-Modal Retrieval: For example, retrieving text that complements the visual data (e.g., pulling related descriptions or explanations for a given image).

2. Multimodal Input Handling:
    - Textual Input: The system must accept text-based queries, which can be general questions or specific tasks.
    - Visual Input: The system should accept images or videos for processing, especially when the input is a combination of text and visual data (e.g., Visual Question Answering or image captioning).
    - Audio Input: If applicable, the system should process audio input (e.g., speech or sound) and convert it into text (e.g., speech recognition).
    - Mixed Modal Input: The system should handle mixed input, where a user may provide both text and images or other combinations of modalities.

3. Multimodal Fusion:
    - The system should merge the retrieved information from different modalities (e.g., text and images) into a unified representation. This requires designing a fusion mechanism to combine these different types of data effectively.
    - Fusion Strategies: Early fusion (combining raw modalities at the input level), late fusion (merging outputs from separate modalities), or hybrid fusion (a combination of both).
4. Text Generation:

- The system must generate natural language text based on the retrieved information. This could include generating responses, captions, summaries, or answers to questions.
- The generated text should incorporate information from the relevant multimodal data (e.g., a text response informed by both a visual input and a text document).

5. Query Understanding and Intent Detection:
   - The system needs to interpret the user's intent, especially when the input involves multimodal data. This includes detecting the type of task (e.g., VQA, image captioning, etc.) and the specific question or request the user is making.
   - Natural Language Understanding (NLU): Recognizing intent, extracting entities, and understanding the context of multimodal queries.
6. Output Generation:
   - Based on the retrieved and fused data, the system should generate an output in the form of text (e.g., an answer to a question, a description, or a summary).
   - If needed, the output could also be multimodal (e.g., generating text descriptions for images or videos, or suggesting actions based on multimodal input).
7. Feedback and Learning:
   - The system may include a feedback loop, where user interaction with the output (e.g., accepting or rejecting answers) is used to improve future retrieval and generation performance.

2. Non-Functional Requirements

These define the performance characteristics of the system, focusing on reliability, efficiency, scalability, and other quality attributes.

Key Non-Functional Requirements:

1. Scalability:
   - The system should be able to scale as the amount of multimodal data (text, images, videos, etc.) grows. Efficient storage and retrieval mechanisms need to be in place to handle large volumes of data.
   - Retrieval mechanisms should be optimized for fast querying, especially when dealing with vast multimodal datasets.

2. Performance:
   - The system should provide low-latency responses, particularly in real-time applications like chatbots or customer support systems.
   - The model's inference time should be optimized for each modality, especially when dealing with complex models like multimodal transformers.

3. Accuracy:
   - The system must accurately retrieve and generate responses. Retrieval should provide highly relevant data from the appropriate modalities, and the generated outputs should be coherent and relevant.
   - Evaluation metrics like BLEU, ROUGE, or accuracy should be employed to assess the quality of the output in tasks like captioning, question answering, etc.

4. Robustness:

   - The system should handle noisy or incomplete inputs gracefully (e.g., images with low resolution, ambiguous text, or audio with background noise).

- Multimodal fusion must be resilient to incomplete or missing data from any one modality, ensuring that the system can still generate useful outputs.

5. Security and Privacy:
   - Data should be protected, especially if the system handles sensitive multimodal information (e.g., medical images or private documents).
   - The system must comply with data privacy laws (e.g., GDPR, HIPAA) and ensure secure handling of multimodal data.
6. Usability:
   - The system should provide a user-friendly interface, allowing easy interaction with both textual and multimodal inputs. Users should be able to easily submit multimodal queries and receive multimodal outputs in a clear format.
   - The response format should be adaptable to different types of outputs (e.g., showing images with captions or providing text summaries with video links).
7. Extensibility:
   - The system should be designed to easily incorporate new modalities or additional data sources as needed. For example, adding a new video source or integrating audio understanding can be done with minimal changes to the core architecture.
   - The framework should allow future expansion of the retrieval and generation components to support emerging tasks or domain-specific use cases.
8. Interpretability and Explainability:
   - Users should be able to understand why the system made a certain decision or provided a specific output. For example, in a medical context, the system should explain why it provided a particular diagnosis or recommendation, and it should reference the modalities involved in the decision (e.g., "Based on the X-ray image and the patient's symptoms, this is the likely diagnosis").
   - Tools for visualizing multimodal data relationships (e.g., attention maps over images or videos) should be considered for improving transparency.

4. Data Requirements

For a Multimodal RAG system to work effectively, it must access a variety of multimodal datasets, including both structured and unstructured data sources.

Key Data Requirements:

1. Multimodal Datasets:
   - Textual Data: Large-scale textual corpora relevant to the domain, such as documents, articles, medical records, FAQs, or manuals.
   - Image/Video Data: Images or video datasets, such as COCO (for image captioning), YouTube-8M (for video recognition), or medical image datasets (e.g., X-rays, MRIs) for healthcare applications.
   - Audio Data: Speech datasets if the system needs to process spoken language, such as LibriSpeech or CommonVoice.
   - Annotations: Datasets with labeled annotations that map text to images (e.g., captions for images), or question-answer pairs for VQA tasks.
2. Data Storage:
   - A storage system that can efficiently handle multimodal data, possibly distributed across different storage solutions (e.g., databases for text, object storage for images and videos).

- Ensure that data can be indexed and retrieved quickly based on its modality and features.

3. Data Preprocessing:
   - Text should be tokenized and processed into embeddings (using models like BERT or GPT).
   - Images and videos need to be processed into feature vectors using deep learning models (e.g., CNNs, Vision Transformers).
   - Audio data should be transcribed into text and processed using speech recognition models.

5. Technical Requirements

This involves identifying the hardware, software, and infrastructure necessary to develop and deploy the Multimodal RAG system.

Key Technical Requirements:

1. Hardware:
   - High-performance computing resources, especially for training multimodal models (e.g., GPUs or TPUs).
   - Sufficient storage for large multimodal datasets and model checkpoints.
2. Software/Frameworks:
   - Deep learning libraries like TensorFlow, PyTorch, or Hugging Face for training and fine-tuning multimodal models.
   - Transformers library for implementing state-of-the-art RAG or multimodal architectures.
   - Elasticsearch or similar tools for efficient multimodal retrieval.
   - Pretrained models (e.g., CLIP, ViT, BLIP) for multimodal understanding.
3. Infrastructure:
   - Cloud platforms (e.g., AWS, Google Cloud, or Azure) for scalable storage and compute resources, especially for training large-scale multimodal models.
   - Data pipeline management systems for handling large volumes of multimodal data (e.g., Apache Kafka, Apache Spark).

6. Library Module

Redis

Redis (Remote Dictionary Server) is an in-memory data store often used as a cache, database, or message broker. Redis is highly performant and can serve as a vector database to store and retrieve embeddings efficiently.

Key Features:

- Caching – Speeds up applications by caching frequently accessed data.
- Vector Database – Stores embeddings for similarity search (common in LLM applications).
- Real-time Data – Handles real-time analytics, session storage, and rate limiting.
- Scalable – Easily scalable and distributed.

Use Cases:

- Caching AI results, storing embeddings for RAG, and fast data retrieval.

ChromaDB

ChromaDB is an open-source vector database designed specifically for AI applications. It stores and retrieves high-dimensional embeddings, making it ideal for retrieval-augmented generation (RAG) and similarity search.

Key Features:

- Vector Search – Quickly finds data points close to a given embedding.
- Simple API – Easy to use for developers building AI apps.
- Scalable – Handles large datasets of embeddings.
- AI Focused – Built to enhance LLM-based applications by efficiently managing context and knowledge.

Use Cases:

- Semantic search, document retrieval, recommendation engines, and chatbots with memory.

How They Work Together:

- LangChain orchestrates LLM workflows and integrates with vector databases like Redis or ChromaDB to enhance memory and retrieval.
- Google Cloud AI Platform hosts and trains models, allowing LangChain apps to run on scalable infrastructure.
- Redis or ChromaDB store embeddings and documents, enabling fast, context-aware responses in LLM applications.

Example Workflow:

- A user query is processed by LangChain.
- LangChain retrieves relevant data from ChromaDB or Redis (using embeddings).
- The enriched prompt is sent to an LLM hosted on Google Cloud AI Platform (Vertex AI).

- The response is returned to the user with contextual knowledge.

This combination is powerful for building scalable, intelligent AI-driven applications.

Streamlit :

Streamlit is an open-source Python library used to create interactive and visually appealing web applications for data science and machine learning projects with minimal effort. It simplifies the process of turning Python scripts into shareable web apps without needing extensive knowledge of web development (like HTML, CSS, or JavaScript).

Key Features of Streamlit :

- Ease of Use: Write a Python script, and with just a few lines of code, create an interactive dashboard or app.

- Widgets: Provides intuitive widgets like sliders, buttons, and text inputs to make apps interactive. Widgets update in real-time as users interact with them.
- Data Visualization: Supports integration with popular Python libraries like Matplotlib, Plotly, Altair, and more to create rich visualizations.
- Deployment: Streamlit apps can be deployed easily to platforms like Streamlit Community Cloud, Heroku, AWS, etc.
- Live Code Reloading: Apps update automatically whenever you save your Python script.

Common Use Cases:

- Data Dashboards
- Machine Learning Model Demonstrations
- Interactive Reports
- Prototyping Web Apps for Data Projects

LangChain

LangChain is an open-source framework that helps developers build applications using large language models (LLMs). It simplifies combining LLMs with external data, APIs, and custom logic to create advanced, stateful applications.

Key Features:

LLM Integration – Connects to models like OpenAI's GPT, Google Gemini, and more.

Chaining – Links multiple LLM calls to handle complex workflows.

Memory – Enables context retention for chatbots and interactive tools.

Agents – Uses LLMs to autonomously interact with APIs and tools.

Data Access – Integrates with vector databases (like Redis and ChromaDB) for retrieval-augmented generation (RAG).

Use Cases:

Chatbots, virtual assistants, document search, and question-answering systems.

LangChain-Chroma :-

LangChain-Chromai s an integration between LangChain and Chroma, a vector database designed for storing and querying embeddings. This combination allows developers to build applications that can efficiently search and retrieve relevant information from large datasets by leveraging vector search technology.

How It Works:
Chroma stores text or data in the form of vector embeddings.

- LangChain can generate embeddings using large language models (LLMs) like OpenAI, and then store, search, and retrieve data from Chroma.

- This enables semantic search, similarity matching, and retrieval-augmented generation (RAG), enhancing the ability of LLMs to find contextually relevant data.

Key Features of LangChain-Chroma:

1. Efficient Vector Search – Enables fast and scalable similarity searches across large datasets.
2. Seamless Integration – LangChain makes it easy to integrate Chroma as a storage and retrieval backend.
3. Memory for LLMs – Chroma can be used as a long-term memory for language models, storing conversation history or knowledge bases.
4. Real-time Retrieval – Enables models to access and query data dynamically during interaction, improving response accuracy and relevance.

Common Use Cases:

- Question Answering Systems – Retrieve relevant documents from large databases.
- Chatbots with Context – Store and retrieve past conversations or relevant knowledge.
- Document Search Engines – Perform semantic searches across unstructured data.
- AI-Powered Research Tools – Search for similar research papers, articles, or documents by meaning rather than keywords.

Tesseract-OCR :-

Tesseract-OCR is an open-source Optical Character Recognition (OCR) engine developed by Google. It is widely used to extract text from images, scanned documents, and PDFs, enabling the conversion of non-editable text into machine-readable formats.

Key Features :

- Multi-Language Support – Recognizes text in over 100 languages.
- High Accuracy – Provides reliable OCR results, especially when paired with image preprocessing.
- Flexible Input – Works with various image formats (JPEG, PNG, TIFF, etc.).
- PDF Extraction – Can extract text from scanned PDFs by integrating with tools like Poppler.
- Extensible – Supports custom language training and fine-

  tuning. How Tesseract Works :

1. Preprocessing – Enhances image quality (e.g., noise reduction, resizing).
2. Text Detection – Identifies areas of the image containing text.
3. Character Recognition – Converts detected text into machine-readable format.
4. Output – Produces text output (plain text, searchable PDFs, etc.).

Why Use Tesseract-OCR?

- Free and Open Source – No licensing fees, making it cost-effective.
- Customizable – Train Tesseract to recognize specific fonts or character sets.
- Integration – Easily integrates with Python, command line tools, and other software.
- Document Digitization – Ideal for converting scanned books, forms, and receipts into editable text.

LangChain-Google-GenAI :-

LangChain-Google-GenAI is an integration between LangChain and Google's Generative AI models (such as Gemini and PaLM). It allows developers to leverage Google's advanced large language models (LLMs) within the LangChain framework to build AI-powered applications.

This integration simplifies connecting to Google's LLM APIs and enables seamless incorporation of generative AI capabilities into applications for tasks like text generation, summarization, and more.

Key Features :

1. Access to Google AI Models – Direct access to Google's Gemini and PaLM models for text and multimodal generation.
2. Seamless Integration – Easily plug Google's models into LangChain workflows and agents.
3. Scalable – Use Google's AI infrastructure for high-performance, scalable applications.
4. Multimodal Support – Access models that handle text, images, and structured data.

Use of LangChain-Google-GenAI :
. State-of-the-Art Models – Google's LLMs (like Gemini) offer top-tier performance for AI tasks.
. Simple API Integration – Reduces complexity when calling Google's generative models.
. Versatility* – Enables complex workflows like retrieval-augmented generation (RAG) using Google models.
. Cloud Deployment – Benefit from Google Cloud's reliability and scalability.

Unstructured[all-docs] :-

unstructured[all-docs] is part of the Unstructured Python library, which is designed to extract and process content from a wide range of unstructured documents (like PDFs, HTML, Word files, images, and more).
The [all-docs] extension is a dependency group that ensures the installation of all the necessary packages and tools required to handle all document types supported by the library.

Key Points :

- unstructured : A library that processes and extracts text and data from unstructured documents.

- [all-docs] : Installs support for extracting content from PDFs, Word documents, HTML files, images, spreadsheets, and more.

Why Use unstructured[all-docs]:

- Broad Document Support – Ensures compatibility with all major document types.

- Ease of Use – One command installs everything needed to handle multiple formats.
- Versatility – Supports mixed data pipelines where documents vary in type and structure.

Common Document Types Supported:

- PDFs
- HTML and XML files
- Word (DOCX)
- Excel (XLSX)
- PowerPoint (PPTX)
- Images (with OCR via tools like Tesseract)
- Text files (TXT, CSV)

htmltabletomd :-

htmltabletomd is a Python library that converts HTML tables into Markdown format. It simplifies the process of extracting and transforming tabular data from HTML documents or web pages into a clean, readable markdown format, which is often used for documentation, blogs, and static sites.

Key Features :

- HTML to Markdown – Converts complex HTML tables into properly formatted Markdown tables.
- Lightweight and Simple – Focused specifically on table conversion, making it fast and easy to use.
- Automation Friendly – Can be integrated into scripts or pipelines that process HTML files for documentation or data extraction.

Why Use htmltabletomd:

- Markdown Readability – HTML tables can be cumbersome in markdown files. This tool automates the conversion, ensuring neat and consistent formatting.
- Documentation – Useful for generating markdown-based documentation from HTML tables.
- Web Scraping – When extracting tabular data from websites, htmltabletomd can convert that data for easy integration into markdown reports.

Use Cases :

·Converting HTML to Markdown - for technical blogs, documentation, or GitHub READMEs.

- Extracting Tables from Web Data - and transforming them for use in markdown reports.

- Automated Reporting – Converting HTML tables from web apps or emails into markdown reports.

Poppler-Utils :-

Poppler-Utils is a collection of command-line tools used for processing and manipulating PDF files. It is part of the Poppler PDF rendering library, which is an open-source implementation of Adobe's PDF rendering. These utilities provide a lightweight, efficient way to extract text, convert PDFs to images, and manage PDF metadata.
Use Cases :

- PDF to Text Conversion – Extract readable text for document analysis.
- Image Extraction – Pull images from PDFs for reuse or processing.
- Metadata Analysis – Automate metadata retrieval for PDF archives.
- Web Conversion – Turn PDFs into HTML for web publication.

Poppler-Utils is a powerful, open-source toolkit that simplifies PDF manipulation, making it indispensable for developers, researchers, and anyone working with PDF documents.

Pydantic :-

Pydantic is a Python library for data validation and parsing. It uses Python type hints to validate input data, ensuring that your data structures are correct and well-formed. Pydantic is widely used in applications like FastAPI for API data validation, configuration management, and parsing JSON data into Python objects.

Key Features of Pydantic :

1. Data Validation – Automatically validates input data based on type hints.
2. Parsing – Converts data from various formats (JSON, dictionaries) into Python objects.
3. Error Handling – Provides detailed and easy-to-understand error messages when data validation fails.
4. Performance – Written in Python with performance-critical parts in C, making it fast.
5. JSON Schema Generation – Can generate JSON Schema definitions directly from Pydantic models.
6. Nested Models – Supports complex data structures with nested models.

Use of Pydantic :

. Reliability – Ensures data is correctly structured before it enters your application logic.

. Simplicity – Reduces boilerplate code for data validation.

. Integration – Works seamlessly with modern frameworks like FastAPI and LangChain .

. Security – Validates user input, reducing the risk of processing invalid or harmful data.

Use Cases :

. API Development – Validating request and response bodies in FastAPI.

. Configuration Management – Validating environment variables and app configurations.

. Data Pipelines – Ensuring incoming data is clean and structured.

. Form Validation – Validating user input in web forms or CLI tools.

Chapter-5: System Flow

Multimodal RAG System Flow

In this section, we will break down the system flow for a Multimodal Retrieval-Augmented Generation (RAG) system using various diagram types. These diagrams help visually represent the interactions between components and data, as well as the control flow through the system. We will include the following diagrams:

1. Use Case Diagram
2. Data Flow Diagram (DFD)
3. Control Flow Diagram (CFD)

1. Use Case Diagram:

The workflow illustrated above will first use a document parsing tool like Unstructured to extract the text, table and image elements separately. Then we will pass each extracted element into an LLM and generate a detailed text summary as depicted above. Next we will store the summaries and their embeddings into a vector database by using any popular embedder model like OpenAI Embedders. We will also store the corresponding raw document element (text, table, image) for each summary in a document store, which can be any database platform like Redis. The multi-vector retriever links each summary and its embedding to the original document's raw element (text, table, image) using a common document identifier (doc_id). Now, when a user question comes in, first, the multi-vector retriever retrieves the relevant summaries, which are similar to the question in terms of semantic (embedding) similarity, and then using the common doc_ids, the original text, table and image elements are returned back which are further passed on to the RAG system's LLM as the context to answer the user question.

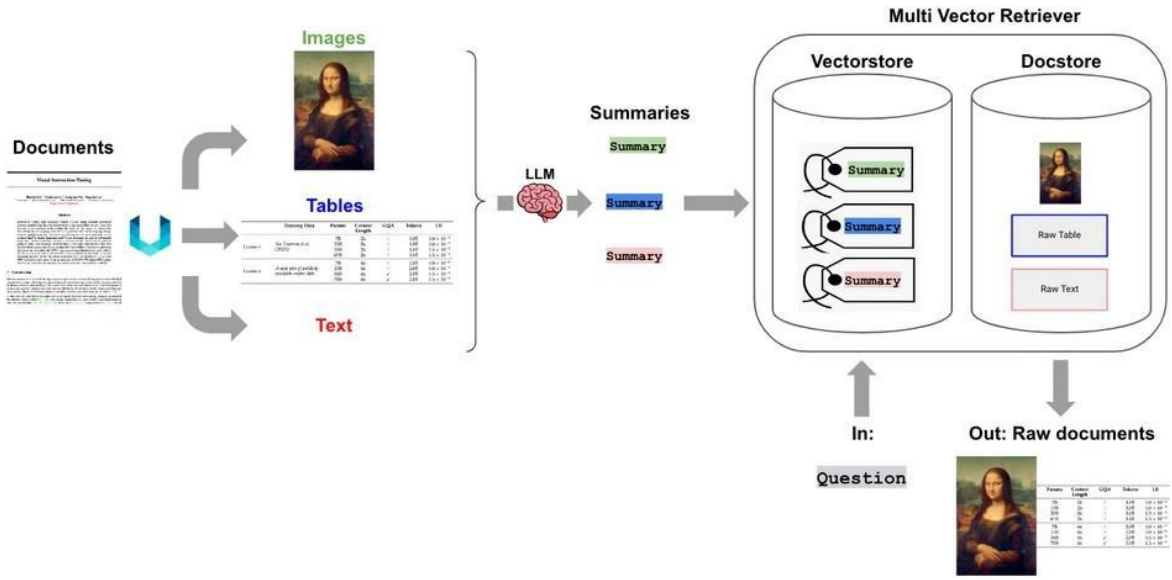

Diagram 1:- Use Case Diagram

2. Data Flow Diagram (DFD):

Data Processing and Indexing, we focus on getting our custom enterprise data into a more consumable format by loading typically the text content from these documents, splitting large text elements into smaller chunks, converting them into embeddings using an embedder model and then storing these chunks and embeddings into a vector database as depicted in the following figure.

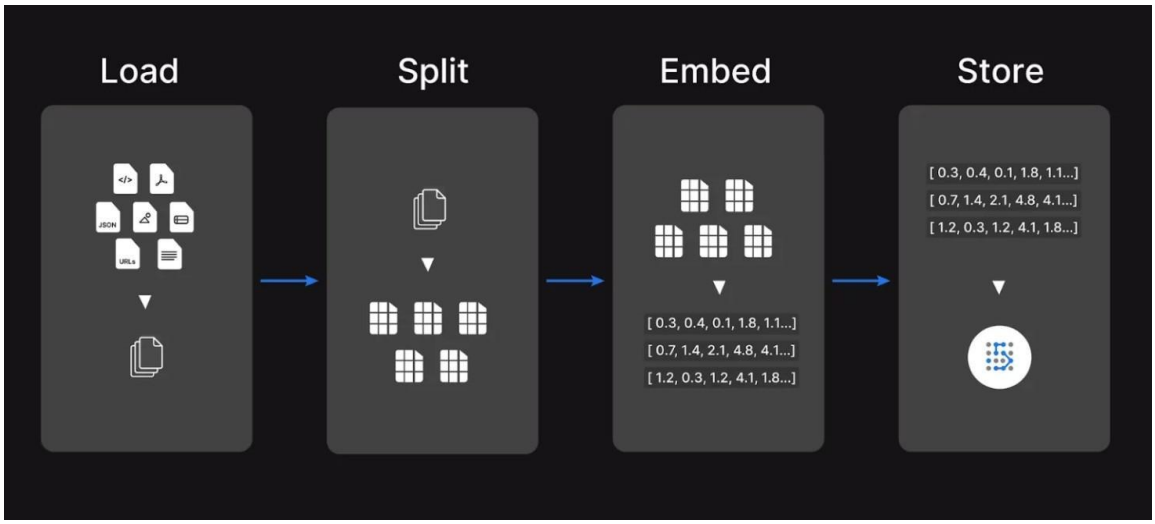

Diagram 2:-Data Flow Diagram

3. Control Flow Diagram (CFD)

The Control Flow Diagram (CFD) describes the control logic and sequence of operations involved in processing the multimodal query and generating the response. It outlines the decision-making and control flow of the system components.

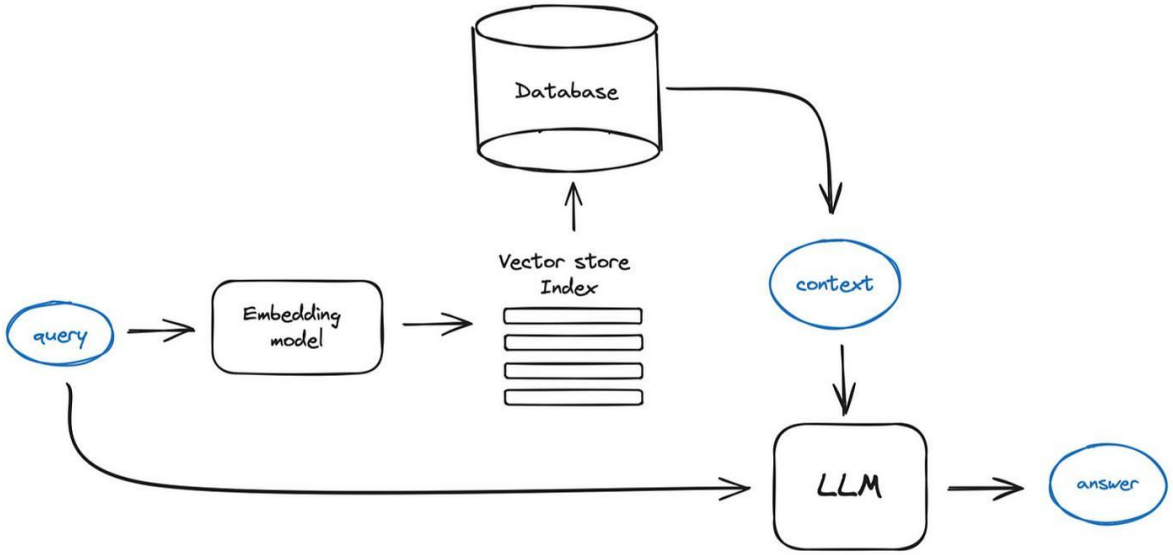

Diagram 3:- Control Flow Diagram

Chapter-6: Proposed Design

Proposed Design of Multimodal Retrieval-Augmented Generation (RAG) System

A Multimodal Retrieval-Augmented Generation (RAG) system integrates various modalities (e.g., text, images, audio) to enhance the quality and relevance of generated responses. The goal is to leverage both retrieved data and generation models to produce contextually aware and informative outputs. Here's a detailed breakdown of a proposed design for such a system.

1. System Architecture Overview
    1. Load all documents and use a document loader like unstructured.io to extract text chunks, image, and tables.
    2. If necessary, convert HTML tables to markdown; they are often very effective with LLMs
    3. Pass each text chunk, image, and table into a multimodal LLM like GPT-4o and get a detailed summary.

    4. Store summaries in a vector DB and the raw document pieces in a document DB like Redis.
    5. Connect the two databases with a common document_id using a multi-vector retriever to identify which summary maps to which raw document piece.

    6. Connect this multi-vector retrieval system with a multimodal LLM like GPT-4o.
    7. Query the system, and based on similar summaries to the query, get the raw document pieces, including tables and images, as the context.

    8. Using the above context, generate a response using the multimodal LLM for the question.

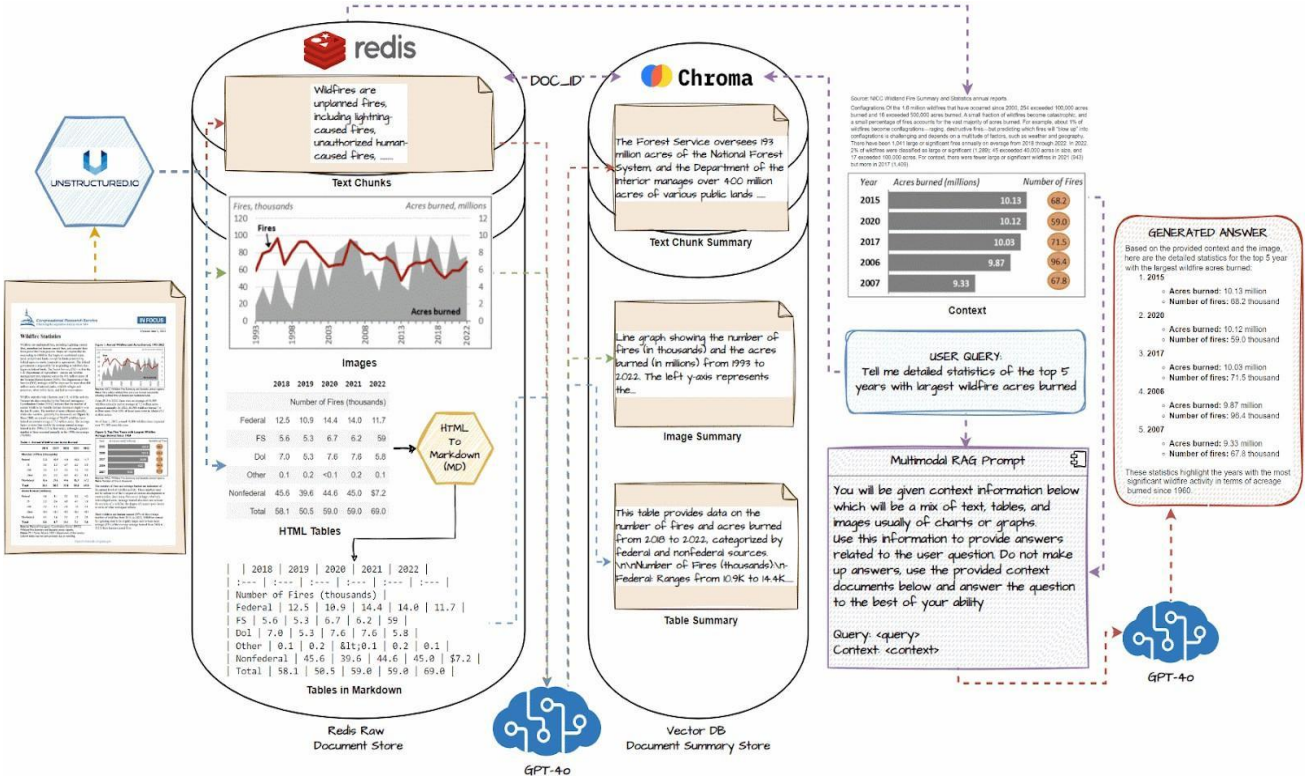

Diagram 4:- System Architecture Diagram

2. Detailed Breakdown of the Components

1. User Interface

- Input Types: The system accepts text, image, or audio as input, allowing multimodal queries.
- Output Types: The system outputs text, images, or audio-based responses. For example, in Visual Question Answering (VQA), the response might be text, while in Image Captioning, it could be a caption alongside an image.
- User Feedback: A feedback loop is included for refining model responses over time. This feedback can be in the form of text-based ratings, corrections, or additional multimodal inputs.

Multimodal Input Handling

- Text Processing: Text input is tokenized and passed through a pre-trained language model (e.g., BERT, GPT, or T5) to extract semantic meaning.
- Image Processing: Image input is processed using Convolutional Neural Networks (CNNs) or Vision Transformers (ViTs) to extract visual features. A model like CLIP can be used to connect image and text embeddings.
- Audio Processing: Audio data is converted into text using Speech-to-Text models (e.g., Whisper, DeepSpeech), or audio features can be directly extracted for analysis.
- Modality Detection: The system first identifies the modality of the input and selects the appropriate processing technique for each (text, image, or audio).

Query Preprocessing

- After the user submits the query, the Query Preprocessing Module normalizes and formats the data from each modality into a unified representation that can be fed into the retrieval and generation modules.
- For audio inputs, the preprocessing converts the audio to text. For images, the preprocessing involves feature extraction (e.g., through a pre-trained CNN like ResNet or Inception).

Multimodal Retrieval

The Multimodal Retrieval component retrieves relevant information from different data sources based on the user query:

- Text Retrieval: This involves querying a text corpus (e.g., Wikipedia, domain-specific datasets) using retrieval models like BM25, dense retrieval (via BERT-based retrievers), or retrieval-augmented generation models.
- Image Retrieval: When the query involves images, the system searches a visual dataset (e.g., ImageNet, COCO, Flickr30k) using content-based image retrieval models like CLIP or Deep Visual Semantic Embeddings (DVSE).
- Audio Retrieval: In case of audio-based queries, the system retrieves relevant audio clips or transcribed text from an audio database (e.g., LibriSpeech, CommonVoice).

Each modality employs a different retrieval mechanism, but the goal is the same: retrieving relevant information to answer the user's query.

5. Multimodal Fusion Model

After retrieving the relevant data, the Multimodal Fusion component is responsible for merging the different modalities (text, image, audio) into a unified representation. Fusion strategies could include:

- Early Fusion: Merge raw features from text, images, and audio before feeding them into a generation model.
- Late Fusion: Process each modality separately and merge the outputs during response generation.
- Cross-Modal Attention: Models like CLIP or VisualBERT can be used to apply attention mechanisms across different modalities, allowing the system to weigh the importance of text and visual elements in the final response.

6. Text Generation Model

The Text Generation model is responsible for generating the final output. This model can be based on transformer architectures such as GPT, T5, or BART, which are fine-tuned for multimodal tasks. The model:

- Takes the fused multimodal representation as input.
- Generates a coherent response based on the context provided by the retrieved data.

For tasks like VQA (Visual Question Answering) or Image Captioning, the generated output might be a text-based answer or description. For Text-to-Image Generation or Image-to-Text Generation, the output could include both textual explanations and relevant visual content.

7. Response Output

The generated response can include:

- Text-based responses (e.g., answering a question or generating a summary).
- Image-based responses (e.g., generated captions for images).
- Audio-based responses (e.g., synthesized speech output).

For applications like Image Captioning, the system may also return the image with the caption. For multimodal queries, such as combining audio and text, the output might include both transcribed text and visual context.

8. User Feedback Loop

To improve system performance, User Feedback is integrated into the system:

- Explicit Feedback: Users can rate the system's responses or provide corrections (e.g., "This answer is correct" or "This caption is inaccurate").
- Implicit Feedback: User behavior, like click-through rates or the time spent on a particular answer, can also be used to refine the system's future responses.

Feedback data is used to fine-tune the retrieval and generation models, as well as to improve the fusion mechanism over time.

3. Design Flow of the System

Step 1: User Input

- The user provides a multimodal query: text, image, or audio.

Step 2: Query Preprocessing

- The system processes the input data, converting images to feature representations, audio to text (if needed), and tokenizing any text input.

Step 3: Multimodal Retrieval

- The system performs separate retrieval for text, image, and audio data using relevant models and data sources.

Step 4: Fusion of Retrieved Data

- The system fuses the multimodal data into a unified representation using fusion mechanisms such as cross-modal attention or early/late fusion.

Step 5: Text Generation

- A text generation model processes the fused multimodal data to generate a coherent response, which could be in the form of text, image captions, or audio descriptions.

Step 6: Response Output

- The response is provided to the user, with multimodal outputs as appropriate (text, image, audio).

Step 7: User Feedback

- Users provide feedback that helps fine-tune the retrieval and generation models for better performance in future queries.

4. Scalability & Deployment Considerations

To ensure that the Multimodal RAG system is scalable and performs efficiently at scale, consider the following:

- Cloud Infrastructure: Utilize cloud services like AWS, GCP, or Azure for scalable storage (e.g., S3 buckets for images and videos, databases for text) and compute (e.g., GPU-based instances for model inference).
- Microservices Architecture: Break down the system into independent microservices for each modality (text, image, audio) and service (retrieval, fusion, generation) to ensure modularity and scalability.
- Data Caching: Use caching mechanisms to store frequently retrieved or generated content to reduce latency and improve system speed.
- Model Fine-Tuning: Regularly fine-tune the models based on user feedback and new data to maintain high accuracy and performance.

Chapter-7: Sample core code

```
loader = UnstructuredPDFLoader(file_path=doc,
    strategy='hi_res', extract_images_in_pdf=True,
    infer_table_structure=True,
        chunking_strategy="by_title", max_characters=4000,
        # max size of chunks new_after_n_chars=4000, #
        preferred size of chunks
     combine_text_under_n_chars=2000,
        mode='elements',
      image_output_dir_path='./figures')
      data = loader.load()
      docs = []
      tables = []
      for doc in data:

        if doc.metadata['category'] == 'Table':

          tables.append(doc)

      elif doc.metadata['category'] == 'CompositeElement':
        docs.append(doc)
      for table in tables:

        table.page_content = htmltabletomd.convert_table(table.metadata['text_as_html'])
      os.environ["GOOGLE_API_KEY"] = "AIzaSyDUu5iULD5Kt54IMS31r6z3cIvNjqlD" genai.configure()
      model = ChatGoogleGenerativeAI(model="gemini-pro") prompt
      = ChatPromptTemplate.from_template(prompt_text)
      summarize_chain = (
                {"element": RunnablePassthrough()}

                  |
                prompt
                  |
                model
                |

                StrOutputParser() # extracts response as text

      )

      text_summaries = []
      table_summaries = []
      text_docs = [doc.page_content for doc in docs]
      table_docs = [table.page_content for table in tables]
      text_summaries = summarize_chain.batch(text_docs, {"max_concurrency": 5})
      table_summaries = summarize_chain.batch(table_docs, {"max_concurrency": 5})
 # create a function to encode images def
    encode_image(image_path):
          """Getting the base64 string"""

          with open(image_path, "rb") as image_file:

             return base64.b64encode(image_file.read()).decode("utf-8")

 # create a function to summarize the image by passing a prompt to GPT-4o def
    image_summarize(img_base64, prompt):
```

```
    """Make image summary"""
    model = ChatGoogleGenerativeAI(model="gemini-1.5-flash")
    msg = model.invoke(
      [
      HumanMessage( content=[
            {"type": "text", "text": prompt},
            {
              "type": "image_url",
              "image_url": {"url":
                    f"{img_base64}"},
            },
          ]
        )
      ]
    )
    return msg.content
def generate_img_summaries(path):
  """
    Generate summaries and base64 encoded strings for images
    path: Path to list of .jpg files extracted by Unstructured
    """
# Store base64 encoded images img_base64_list
  = []
# Store image summaries
  image_summaries = []
  # Apply to images
    for img_file in sorted(os.listdir(path)):
      if img_file.endswith(".jpg"):
        img_path = os.path.join(path, img_file)
        base64_image = encode_image(img_path)
        img_base64_list.append(base64_image)
        image_summaries.append(image_summarize(base64_image, prompt))
    return img_base64_list, image_summaries
  # Image summaries
  IMG_PATH = './figures'
  imgs_base64, image_summaries = generate_img_summaries(IMG_PATH)
  os.environ["GOOGLE_API_KEY"] = "AIzaSyDUu5iULD5Kt54IMS31r6z3clyNiqlD"
google_api_key="AIzaSyDUu5iULD5Kt54IMS31r6z3clyNiqlD", model="models/embedding-001",
  def create_multi_vector_retriever(
    docstore, vectorstore, text_summaries, texts, table_summaries, tables,
    image_summaries, images):
    """
```

```
    Create retriever that indexes summaries, but returns raw images or texts
    """
    id_key = "doc_id"

    # Create the multi-vector retriever
    retriever = MultiVectorRetriever(

        vectorstore=vectorstore,
        docstore=docstore,
        id_key=id_key,

    )

# Helper function to add documents to the vectorstore and docstore
  def add_documents(retriever, doc_summaries, doc_contents):
        doc_ids = [str(uuid.uuid4()) for _ in doc_contents]
        summary_docs = [
          Document(page_content=s, metadata={id_key: doc_ids[i]})
          for i, s in enumerate(doc_summaries)
        ]

        retriever.vectorstore.add_documents(summary_docs)
        retriever.docstore.mset(list(zip(doc_ids, doc_contents)))


  # Add texts, tables, and images

# Check that text_summaries is not empty before adding if
  text_summaries:
        add_documents(retriever, text_summaries, texts)

# Check that table_summaries is not empty before adding if
  table_summaries:
        add_documents(retriever, table_summaries, tables)

# Check that image_summaries is not empty before adding if
  image_summaries:
        add_documents(retriever, image_summaries, images)
    return retriever
chroma_db = Chroma( collection_name="mm_rag",
  embedding_function=gemini_embed_model,
  collection_metadata={"hnsw:space": "cosine"},
  persist_directory="./chroma_data"
  )

  client = get_client('redis://localhost:6379')

  redis_store = RedisStore(client=client)
retriever_multi_vector = create_multi_vector_retriever(
redis_store, chroma_db,
    text_summaries, text_docs,
    table_summaries, table_docs,
    image_summaries, imgs_base64,
  )

  def plt_img_base64(img_base64):
```

```
        """Disply base64 encoded string as image"""
        # Decode the base64 string
        img_data = base64.b64decode(img_base64)
        # Create a BytesIO object
        img_buffer = BytesIO(img_data)
        # Open the image using PIL
        img = Image.open(img_buffer)
        display(img)
    def looks_like_base64(sb):

        """Check if the string looks like base64"""

        return re.match("^[A-Za-z0-9+/]+[=]{0,2}$", sb) is not None
# helps in checking if the base64 encoded image is actually an image def
  is_image_data(b64data):
        """

        Check if the base64 data is an image by looking at the start of the data
        """
    image_signatures = { b"\xff\xd8\xff":
        "jpg",
          b"\x89\x50\x4e\x47\x0d\x0a\x1a\x0a": "png",
          b"\x47\x49\x46\x38": "gif",
          b"\x52\x49\x46\x46": "webp",
        }

        try:

          header = base64.b64decode(b64data)[:8] # Decode and get the first 8 bytes for
          sig, format in image_signatures.items():
          if header.startswith(sig): return True
     return False except
        Exception: return
              False
# returns a dictionary separating images and text (with table) elements
  def split_image_text_types(docs):
        """

        Split base64-encoded images and texts (with tables)
        """
        b64_images = []
        texts = []
        for doc in docs:

    # Check if the document is of type Document and extract page_content if so if
        isinstance(doc, Document):
            doc = doc.page_content.decode('utf-8')
          else:
            doc = doc.decode('utf-8')

        if looks_like_base64(doc) and is_image_data(doc):
          b64_images.append(doc)
          else:

            texts.append(doc)
```

```
    return {"images": b64_images, "texts": texts}
  def multimodal_prompt_function(data_dict):
    """
    Create a multimodal prompt with both text and image context.
    This function formats the provided context from `data_dict`, which contains text,
    tables, and base64-encoded images. It joins the text (with table) portions
    and prepares the image(s) in a base64-encoded format to be included in a
    message.
    The formatted text and images (context) along with the user question are used to
    construct a prompt for GPT-4o
    """
    formatted_texts = "\n".join(data_dict["context"]["texts"])
    messages = []
# Adding image(s) to the messages if present if
  data_dict["context"]["images"]:
    for image in data_dict["context"]["images"]: image_message
      = {
          "type": "image_url",
          "image_url": {"url": f"{image}"},
        }
        messages.append(image_message) # Adding the text for analysis
  text_message  =  {
    "type":     "text",
    "text": (
      f"""You are an analyst tasked with understanding detailed information
        and trends from text documents,
          data tables, and charts and graphs in images.
          You will be given context information below which will be a mix of text, tables, and
          images usually of charts or graphs.
          Use this information to provide answers related to the user
          question.
          Do not make up answers, use the provided context documents below and
          answer the question to the best of your ability.
          User question:
          {data_dict['question']}
          Context documents:
          {formatted_texts}
          Answer:"""), }
    messages.append(text_message)
    return [HumanMessage(content=messages)] # Create RAG chain
  model = ChatGoogleGenerativeAI(model="gemini-1.5-flash") multimodal_rag = (
      {
        "context": itemgetter('context'),
        "question": itemgetter('input'),
      }
```

```
            |
        RunnableLambda(multimodal_prompt_function)
            |
        model
            |
        StrOutputParser()
    )

# Pass input query to retriever and get context document elements
    retrieve_docs = (itemgetter('input')
                    |
                retriever_multi_vector
                    |
                RunnableLambda(split_image_text_types))
    multimodal_rag_w_sources = (RunnablePassthrough.assign(context=retrieve_docs)
                                .assign(answer=multimodal_rag))
    def multimodal_rag_qa(query):
        response = multimodal_rag_w_sources.invoke({'input': query})
        st.write('=='*50)
        st.write('Answer:')
        st.markdown(response['answer'])
        st.write('--'*50)
        st.write('Sources:')
        text_sources = response['context']['texts']
        img_sources = response['context']['images']
        for text in text_sources:

            st.markdown(text)
            st.write()
    for img in img_sources:
        plt_img_base64(img)
        st.write()
        st.write('=='*50)
    st.write("Readyfor Answering")
user_input = st.text_input( "Enter a
    you question:",
    placeholder="Type here...",
    max_chars=50)
    if user_input:
    st.write(f"Your Question is: {user_input}")
    st.write()
    multimodal_rag_qa(user_input)
```

Chapter-8: Experimental Result

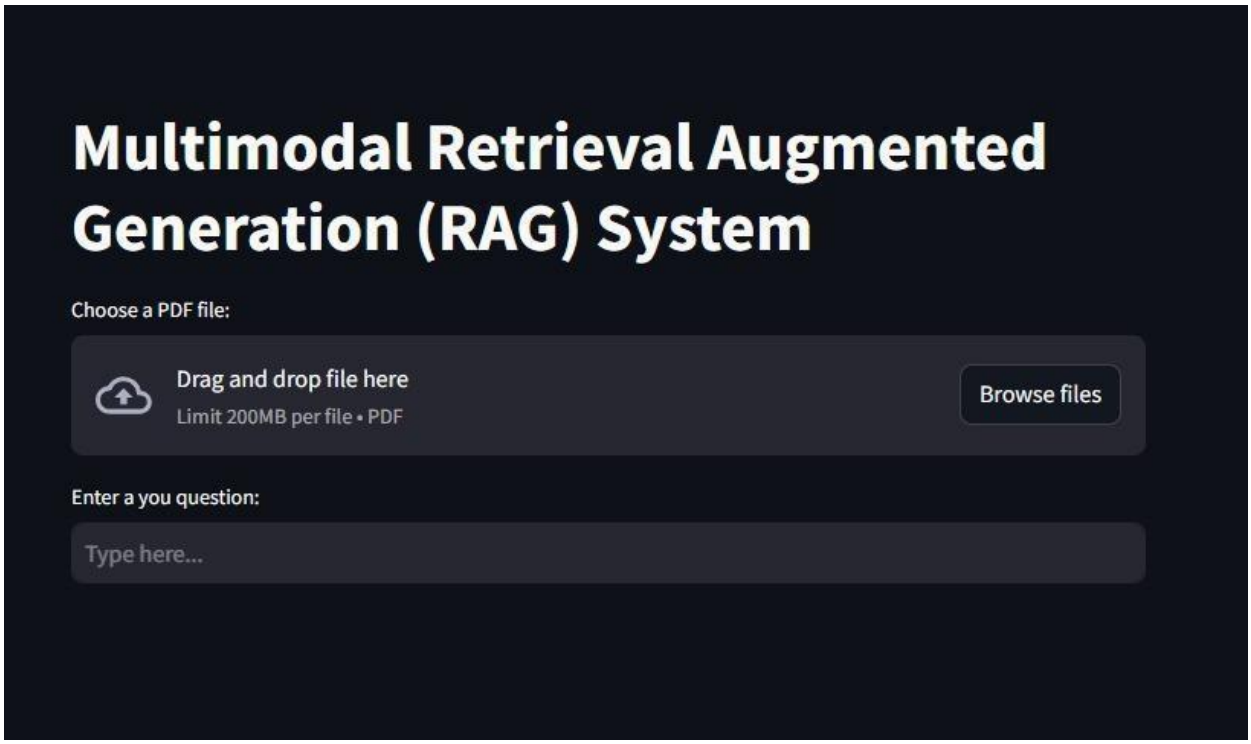

Figure 1:- Output view

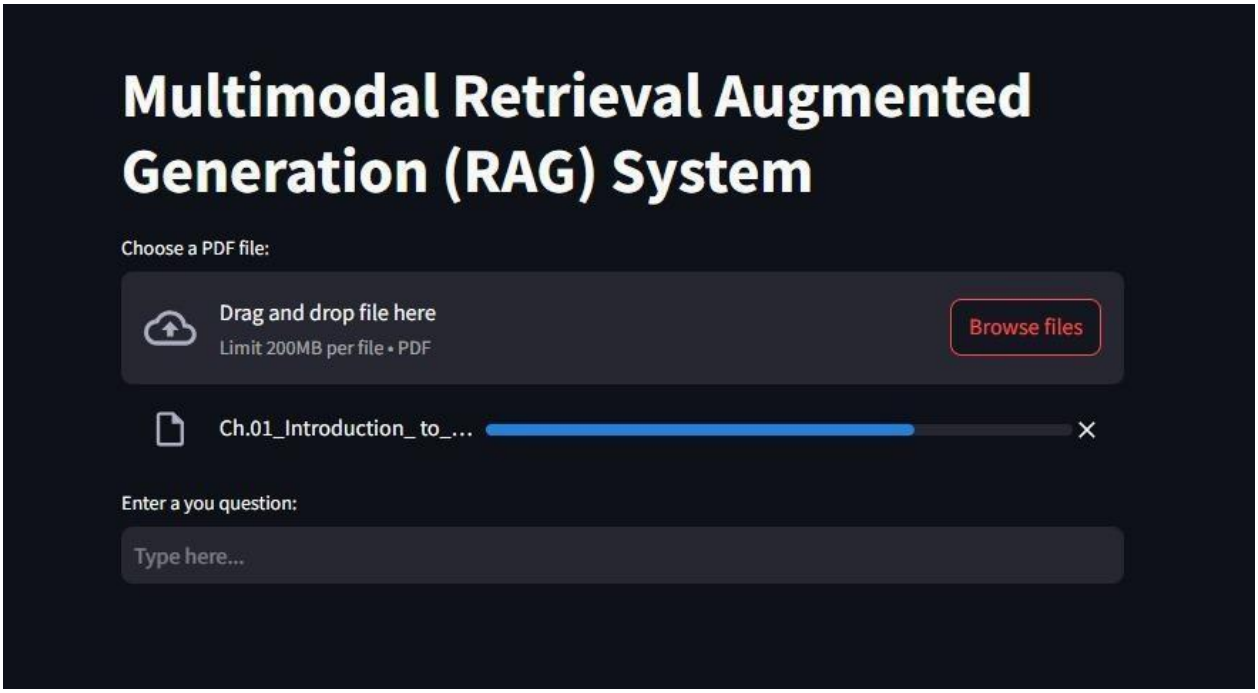

Figure 2:- File Upload

Figure 3:- Answer of a question

Chapter-9: Future Scope

Future Scope of Multimodal Retrieval-Augmented Generation (RAG)

The **Multimodal Retrieval-Augmented Generation (RAG)** system is a cutting-edge paradigm that integrates multiple modalities (text, image, video, audio, etc.) to enhance AI capabilities in natural language understanding and generation. As this field is rapidly evolving, there are several exciting **future directions** for the development and deployment of **Multimodal RAG** systems. Here are some of the most promising areas of future research and application:

1. **Expansion of Modalities**

As of now, **Multimodal RAG** primarily deals with common modalities like **text**, **images**, and **audio**. However, there is substantial potential to integrate additional modalities:

- **Video**: Integrating video content with text, images, and audio can provide even richer context for generation models, useful for tasks like **video captioning**, **action recognition**, and **video summarization**.
- **Sensor Data**: Real-time data from IoT devices (such as motion sensors, environmental sensors, etc.) can be used for more context-aware RAG models. For instance, in smart cities, sensor data can complement images and text for dynamic decision-making.
- **Gestures and Sign Language**: Understanding and generating responses based on human gestures or sign language can be a crucial area for improving **human-computer interaction** (HCI), making systems more inclusive and accessible.

2. **Enhanced Cross-Modal Understanding**

One of the key challenges for **Multimodal RAG** systems is creating a deep **cross-modal understanding**, where different types of data are not just processed independently but are understood in relation to each other:

- **Multimodal Transformers**: Future models may use advanced architectures like **Multimodal Transformers** that allow different modalities to be jointly represented in a single embedding space. This would improve the ability to transfer knowledge across modalities and provide richer, more accurate results.
- **Multimodal Contextualization**: Building models that understand the context across various modalities is key. For example, a system that can correctly interpret a **text query** like "What is the weather like today in this video?" by using both **the video** (showing a weather forecast) and **the location** in the text.

3. **Real-Time Multimodal RAG Systems**

Currently, many **Multimodal RAG** systems face challenges in real-time applications. The future could see systems capable of processing multimodal inputs in real-time across various industries, including:

- **Real-Time Customer Support**: Combining real-time text-based chats, video feeds, and spoken language for live customer support or troubleshooting, where a **virtual assistant** can analyze all these sources and provide a holistic response.
- **Real-Time Translation and Transcription**: Multimodal systems that integrate audio, text, and visual cues can provide real-time **multilingual translations** and transcriptions, even for complex **sign language interpretation**.

- **Autonomous Systems**: Self-driving cars and robots could use multimodal RAG systems to process inputs from **cameras, LiDAR**, and other sensors for real-time decision-making and problem-solving.

4. Improved Multimodal Fusion Models

Fusing data from multiple modalities is one of the most challenging aspects of **Multimodal RAG** systems. Future research may focus on more sophisticated and efficient fusion models:

- **Deep Fusion Models**: Advanced deep learning models may allow for more seamless fusion of modalities, combining feature extraction and decision-making processes in a more integrated fashion.
- **Attention-Based Fusion**: Attention mechanisms (e.g., **cross-attention** or **self-attention**) could evolve to more effectively manage interactions between different modalities. This would allow the model to dynamically choose which modality is more important depending on the query.
- **Multimodal Representations**: Future models could create richer, more compact multimodal representations that can be used for both retrieval and generation without requiring each modality to be processed independently.

Multimodal Knowledge Graphs and Retrieval

Future **Multimodal RAG** systems could integrate **knowledge graphs** that span across different modalities, allowing for richer **semantic understanding** and more effective multimodal retrieval:

- **Knowledge Graph Integration**: Systems could query knowledge graphs that connect data points from different modalities (e.g., text, images, videos) to provide more relevant and contextually rich responses. For instance, a knowledge graph could link images of a place with historical data, news articles, and even **social media content**, making the retrieval process more dynamic and informative.
- **Multimodal Search Engines**: Imagine a search engine that combines image recognition, natural language processing, and contextual reasoning to return results that seamlessly combine text, images, and videos—beyond simple keyword matching.

Personalized and Context-Aware Multimodal RAG

As the **RAG** models evolve, they could become increasingly personalized to the needs and preferences of individual users:

- **Personalized Multimodal Interactions**: Future multimodal RAG systems may leverage user data (such as interaction history, preferences, location, and even physiological states) to tailor responses that align with individual needs. For instance, **recommendation systems** could integrate user preferences across multiple modalities (e.g., preferred reading styles, visual themes in video content).
- **Context-Aware Generation**: Context-awareness is a major area of research. A future **Multimodal RAG** system could better understand user **intent** and adapt its outputs dynamically based on not only the **current query** but also on **historical context**, **ongoing tasks**, or **external environmental factors**.

Ethical and Fairness Considerations in Multimodal RAG

As with all AI systems, **Multimodal RAG** systems need to address **ethical concerns** regarding fairness, privacy, and inclusivity:

- **Bias Mitigation**: Multimodal systems that incorporate images and videos need to be particularly cautious about **bias** (e.g., racial, gender, or cultural biases) present in the datasets used for training. Future systems will need to focus on fairness-aware model training to avoid reinforcing harmful stereotypes or biases.
- **Explainability and Interpretability**: Multimodal RAG systems, particularly in high-stakes applications (e.g., healthcare, autonomous driving), will need to provide **explainable outputs**. Users must be able to understand why a model has made a particular decision or answer.
- **Data Privacy**: The combination of personal data from multiple modalities (e.g., text, voice, video) could raise **privacy concerns**. Future systems must include mechanisms for user control over what data is collected and how it is used.

Integration with Augmented and Virtual Reality (AR/VR)

With the growing interest in **augmented reality (AR)** and **virtual reality (VR)**, **Multimodal RAG** systems could play a central role in shaping user experiences in immersive environments:

- **AR/VR Integration**: For instance, in a **virtual assistant** scenario, a **Multimodal RAG** could pull real-time text, images, and video data from the physical environment (via camera feeds, sensors, etc.) and overlay this information onto a user's AR display. Similarly, users could speak or gesture in real-time to interact with these systems.
- **Immersive Educational Tools**: In education, **virtual learning environments** could use **Multimodal RAG** systems to generate dynamic and contextually appropriate responses, answering both text-based and visual queries in AR/VR settings.

Continuous Learning and Adaptation

One of the most exciting aspects of **future Multimodal RAG** systems is their ability to **learn continuously** from interaction data:

- **Online Learning**: Future systems could be capable of **online learning**, where the model continuously updates and refines its understanding based on real-time user interactions and feedback.
- **Meta-Learning**: The system could also leverage **meta-learning** to quickly adapt to new domains, tasks, or modalities with minimal additional training, improving its ability to generalize across use cases.

Real-World Applications and Industry Adoption

The integration of **Multimodal RAG** will have significant impacts across various industries:

- **Healthcare**: RAG systems could assist doctors in diagnosing patients by combining medical images (e.g., X-rays, MRI scans) with patient records (text) and even audio inputs from consultations.
- **Retail**: In **e-commerce**, RAG systems could generate product recommendations by combining text reviews, images, and user preferences, improving customer experience.
- **Entertainment**: In **gaming** or **movie production**, RAG systems could combine game dialogue, visual scenes, and interactive elements to generate rich narrative content.

Chapter-10: Conclusion

Conclusion of Multimodal Retrieval-Augmented Generation (RAG)

The **Multimodal Retrieval-Augmented Generation (RAG)** system represents a significant advancement in the field of artificial intelligence by combining the strengths of both **retrieval-based models** and **generative models** across multiple modalities. This fusion enables the system to provide richer, more contextually relevant, and informative outputs by leveraging various types of data—such as text, images, audio, and even video—through a unified processing pipeline.

Key takeaways and conclusions from the study and development of **Multimodal RAG** systems include:

1. Enhancing Human-AI Interaction

The ability of **Multimodal RAG** systems to handle and integrate various forms of input—text, images, audio, and more—marks a leap toward **more natural and intuitive human-AI interactions**. This creates opportunities for more seamless and accurate AI applications, such as **virtual assistants**, **chatbots**, **image captioning**, and **cross-modal search engines**. The technology enables richer dialogues with users, supports complex query answering, and provides contextually relevant responses based on multimodal inputs.

2. Leveraging Cross-Modal Knowledge

By combining data from multiple modalities, **Multimodal RAG** systems facilitate a deeper **cross-modal understanding**. For example, understanding the relationship between an image and a text caption, or the context between an audio query and visual content, allows these systems to generate highly accurate and context-sensitive responses. This capability is crucial in fields like **Visual Question Answering (VQA)**, **image captioning**, and **multimodal search**, where the system must combine text, vision, and sometimes audio to provide holistic answers.

3. Retrieval-Augmented Models Enhance Accuracy

The retrieval-augmented approach addresses one of the main limitations of traditional generative models, which often lack real-world knowledge. By retrieving relevant information from large external knowledge bases before generating responses, **Multimodal RAG** systems can draw on up-to-date, detailed, and specific knowledge, improving the **accuracy** and **relevance** of their output. This is particularly beneficial for dynamic, real-time tasks such as **customer support**, **diagnostics**, and **problem-solving**.

4. The Role of Personalization and Adaptation

As **Multimodal RAG** systems evolve, their ability to **personalize** responses based on individual user preferences, past interactions, and contextual factors will significantly enhance their utility. By continuously adapting to users' needs and preferences, these systems can offer tailored experiences, leading to greater user satisfaction and more effective interactions, especially in fields like **e-commerce**, **healthcare**, and **education**.

5. Bridging Modalities to Improve Understanding

The **fusion of different modalities**—text, images, audio—requires sophisticated models that can combine these data sources effectively. **Multimodal RAG** systems that leverage innovative techniques like **cross-modal attention** and **multimodal transformers** can provide a more holistic understanding of the context and relationships across modalities. This will improve tasks such as **real-time translation**, **multimodal search**, and **autonomous systems**, where accurate understanding of diverse input data is crucial.

6. Ethical and Societal Implications

As **Multimodal RAG** systems become more pervasive, it is essential to address the **ethical considerations** surrounding their use. Issues like **bias**, **privacy**, **security**, and **fairness** must be prioritized to ensure that these systems serve all users equitably. This will require responsible data handling, careful design to avoid reinforcement of harmful stereotypes, and transparent decision-making processes.

7. Scalability and Deployment

The integration of multimodal capabilities into real-world systems presents both challenges and opportunities for scalability. **Cloud computing** and **distributed architectures** can support the large-scale deployment of **Multimodal RAG** systems, ensuring that they perform efficiently in high-demand applications. Moreover, the ongoing development of **more efficient models** (e.g., **parameter-efficient transformers** and **knowledge distillation**) will help scale these systems in real-time and across diverse applications, from enterprise solutions to consumer-facing tools.

8. Future Directions and Innovations

The future of **Multimodal RAG** systems looks promising, with several areas of growth and innovation:

- **Expansion into new modalities**, such as gesture recognition, real-time sensor data, and even complex human emotions, will expand the use cases for **Multimodal RAG** in fields like **augmented reality (AR)**, **virtual reality (VR)**, and **human-computer interaction**.
- **Continual learning** and **meta-learning** will allow these systems to adapt to new data, evolving tasks, and personalized interactions without requiring retraining from scratch.
- **Improved cross-modal reasoning** and the ability to handle **longer, more complex multimodal inputs** will lead to even more sophisticated AI systems in areas like **automated content generation**, **personalized education**, and **multimodal analytics**.

Final Thought

Multimodal Retrieval-Augmented Generation systems are poised to revolutionize how AI interacts with the world by enabling richer, more accurate, and context-aware responses across a range of domains. With advancements in **multimodal understanding**, **knowledge retrieval**, and **generation**, these systems will become increasingly powerful tools, enhancing human-computer interactions and contributing to innovative applications across industries. However, alongside these advancements, careful attention to ethical concerns, data privacy, and user trust will be crucial for their responsible deployment.

As technology evolves, the future of **Multimodal RAG** holds immense potential for transforming fields like healthcare, entertainment, education, and beyond—ushering in a new era of intelligent, adaptable, and empathetic AI systems.

Appendices

Multimodal-RAG Test Cases:

These test cases aim to evaluate different aspects of your Multimodal-RAG system:

1. Retrieval Quality:

Relevance:

Text Query, Relevant Image: Provide a text query and ensure the system retrieves an image that is highly relevant to the query.

Image Query, Relevant Text: Provide an image and ensure the system retrieves text that accurately describes or relates to the image.

Combined Query (Text + Image): Provide both a text query and an image and verify that the retrieved content (text and/or images) is relevant to both.

Diversity:

Varying Query Types: Test with different types of queries (e.g., descriptive, comparative, factual) to ensure the system retrieves a diverse set of relevant content.

Multiple Relevant Items: If multiple relevant images or text passages exist, check if the system retrieves a representative sample of them.

Retrieval Speed: Measure the time it takes for the system to retrieve relevant content.

2. Generation Quality:

Accuracy:

Factuality: Ensure the generated responses are factually accurate and consistent with the retrieved content.

Image Grounding: If the response involves images, verify that the generated text correctly refers to and describes the image content.

Coherence and Fluency:

Logical Flow: Check if the generated responses are coherent and logically structured.

Natural Language: Ensure the generated text is fluent and grammatically correct.

Multimodal Integration:

Cross-Modal Understanding: Test if the system can effectively integrate information from different modalities (e.g., generate a text description based on both a text query and a related image).

Image Description Quality: If the system generates image descriptions, evaluate their quality in terms of accuracy, detail, and relevance to the context.

3. Robustness:

Ambiguous Queries: Test with ambiguous or vague queries to see how the system handles uncertainty.

No Relevant Content: Provide queries for which no relevant content exists in the knowledge base and check if the system provides an appropriate response (e.g., "No relevant information found").

Noisy or Corrupted Data: Test with noisy or corrupted images or text to assess the system's resilience.

Multimodal-RAG Case Studies:

Here are some potential case studies to illustrate the application and testing of Multimodal-RAG systems:

1. E-commerce Product Search:

Scenario: A user searches for a product using a combination of text (e.g., "red dress with floral print") and an image of a similar dress.

Test Cases:

Verify that the system retrieves visually similar dresses and relevant product descriptions.

Test with variations in the text query (e.g., "casual red dress," "summer dress with flowers").

Evaluate the accuracy of generated descriptions for the retrieved products.

2. Medical Image Diagnosis Support:

Scenario: A doctor uploads a medical image (e.g., X-ray) and asks a question about it (e.g., "Is there a fracture?").

Test Cases: Ensure the system retrieves relevant medical literature or case studies related to the image.Evaluate the accuracy of the generated diagnosis support based on the image and retrieved information.Test with images of varying quality and complexity.

3. Educational Content Generation:
Scenario: A student provides a text prompt (e.g., "Explain the process of photosynthesis") and an image related to plants.

Test Cases:Verify that the system generates a comprehensive explanation of photosynthesis, incorporating relevant details from the image.Test with different types of images (e.g., diagrams, photos of plants). Evaluate the clarity and educational value of the generated content.

Reference

1. **"Deep Learning for Vision and Language"**
    - Authors: Karan Ahuja, Balaji Krishnamurthy
    - Covers multimodal AI with detailed sections on combining vision and language models.
2. **"Transformers for Natural Language Processing" (2nd Edition)**
    - Author: Denis Rothman
    - Explains the use of transformer models like GPT and BERT, with some coverage of multimodal applications.
3. **"Multimodal Machine Learning: Techniques and Applications"**
    - Authors: Jiebo Luo, Ruiping Wang, et al.
    - Explores techniques and tools for multimodal systems in detail.
4. **"Retrieval-Augmented Generation: Foundations and Applications"**
    - Author: Patrick Lewis (forthcoming book based on the original RAG paper).
5. **"Practical AI with Python: Multimodal Applications"**
    - Author: Noah Gift
    - Focuses on practical implementation of multimodal AI systems, including MM-RAG.