

✓ Introduction

This project examines the current state of lecture capture technology within academic institutions. While widely adopted for recording lectures and providing students with review materials, existing systems often fall short of meeting contemporary learning demands. Their basic functionality struggles to keep pace with evolving pedagogical standards, highlighting a critical need for enhancement through artificial intelligence. This project proposes a robust solution aimed at transforming raw lecture content into an intelligent, queryable knowledge base. This advancement would enable users to pose precise questions about lecture material and receive instant, accurate answers, even when information is embedded within spoken discourse or scanned documents, thereby significantly enriching the learning experience and the utility of archived lectures.

✓ Installing and Importing necessary libraries

```
!apt-get update
!apt-get install tesseract-ocr libtesseract-dev -y
!pip install Pillow pytesseract pdf2image -q
!pip install --upgrade pip
!pip install pydub
!pip install PyPDF2
!pip install ffmpeg-python
!pip install --upgrade git+https://github.com/huggingface/transformers.git accelerate
!pip install langchain_community
!pip install sentence-transformers
!pip install rank_bm25
!pip install transformers accelerate
!pip install faiss-cpu
!sudo apt-get update
!sudo apt-get install poppler-utils

import google.generativeai as genai
import os
import textwrap
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
```

```

from pydub import AudioSegment
import torch
from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor, pipeline
import PyPDF2
from PIL import Image
import pytesseract
from pdf2image import convert_from_path

from langchain.text_splitter import RecursiveCharacterTextSplitter
from sentence_transformers import SentenceTransformer
from rank_bm25 import BM25Okapi
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import faiss
from IPython.display import display, Markdown

```

 Requirement already satisfied: transformers in /usr/local/lib/python3.11/
  Requirement already satisfied: accelerate in /usr/local/lib/python3.11/di
  Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist
  Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local
  Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/d
  Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.
  Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/d
  Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python
  Requirement already satisfied: requests in /usr/local/lib/python3.11/dist
  Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/p
  Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/pytho
  Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/di
  Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3
  Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/l
  Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in /usr/local/lib/pyt
  Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-p
  Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.11/
  Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist
  Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-p
  Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/l
  Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr
  Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/l
  Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/
  Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local
  Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/
  Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/loc
  Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/loc
  Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/loca
  Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib
  Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/l
 Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/lo
 Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11
 Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11
 Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/pytho
 Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.
 Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib

```

Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/pytho
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/pytho
Requirement already satisfied: faiss-cpu in /usr/local/lib/python3.11/dis
Requirement already satisfied: numpy<3.0,>=1.25.0 in /usr/local/lib/pytho
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dis
Hit:1 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204
Hit:2 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InReleas
Hit:3 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:4 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:5 https://r2u.stat.illinois.edu/ubuntu jammy InRelease
Hit:6 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:7 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:8 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRele
Hit:9 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy
Hit:10 https://ppa.launchpadcontent.net/ubuntuugis/ppa/ubuntu jammy InRele
Reading package lists... Done
W: Skipping acquire of configured file 'main/source/Sources' as repositor
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done

```

✓ Audio Processing (MP3 to WAV & Transcription)

The workflow begins by converting the input MP3 audio file ('Chicken Cottage 4.mp3') to a WAV format, a necessary step for compatibility and optimal performance with subsequent audio processing tools.

An automatic speech recognition (ASR) pipeline is then implemented using the pre-trained `washeed/audio-transcribe` model from Hugging Face. This pipeline efficiently loads the model, leveraging GPU acceleration if available, processes the converted WAV audio, and generates a comprehensive text transcript.

(Note - For the purpose of this prototype, 'Chicken Cottage 4.mp3' serves as a simulated lecture recording where a financial report is being summarized.)

```

mp3_file = "/content/Chicken Cottage 4.mp3"
wav_file = "output.wav"

print(f"Converting {mp3_file} to {wav_file}...")
sound = AudioSegment.from_mp3(mp3_file)
sound.export(wav_file, format="wav")
print("Conversion complete.")

```

```
print("Setting up audio transcription pipeline...")
device = "cuda:0" if torch.cuda.is_available() else "cpu"
torch_dtype = torch.float16 if torch.cuda.is_available() else torch.float32

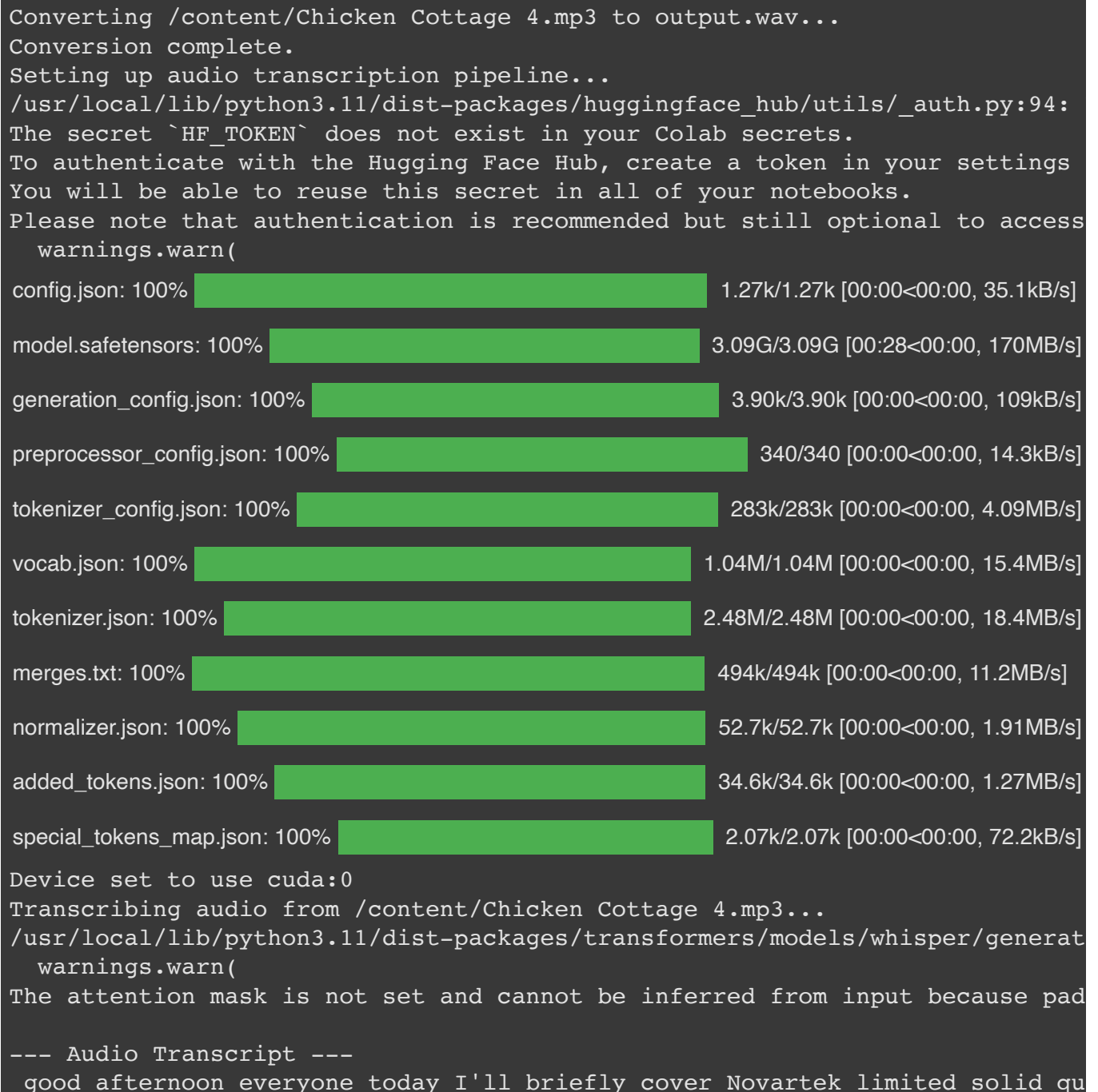
model_id = "washeed/audio-transcribe"

model = AutoModelForSpeechSeq2Seq.from_pretrained(
    model_id, torch_dtype=torch_dtype, low_cpu_mem_usage=True, use_safetensors=
)
model.to(device)

processor = AutoProcessor.from_pretrained(model_id)

pipe = pipeline(
    "automatic-speech-recognition",
    model=model,
    tokenizer=processor.tokenizer,
    feature_extractor=processor.feature_extractor,
    max_new_tokens=128,
    chunk_length_s=30,
    batch_size=16,
    return_timestamps=True,
    torch_dtype=torch_dtype,
    device=device,
)

print(f"Transcribing audio from {mp3_file}...")
result = pipe(mp3_file, generate_kwargs={"task": "transcribe"})
transcript_text = result["text"]
print("\n--- Audio Transcript ---")
print(transcript_text)
```



- ✓ PDF Processing (Text Extraction & OCR for Images)

This section outlines a two-pronged approach for comprehensive text extraction from PDF documents, capturing both native text and image-embedded content.

Initial text extraction is performed by the `read_pdf` function using `PyPDF2`, which directly retrieves searchable text from each page and compiles it into a single string. This step includes error handling for common file-related exceptions.

For text embedded within images (e.g., scanned content), the `extract_text_from_images_in_pdf` function is applied. This involves converting PDF pages into images via `pdf2image`'s `convert_from_path` and then employing `pytesseract` for Optical Character Recognition (OCR) to extract text from these visual representations.

The integration of direct text extraction and OCR ensures a complete capture of all textual data from the PDF, preparing the unified document for subsequent processing steps.

(Note - For the purpose of this prototype, 'NovaTech_Q1_2025_Financial_Report.pdf' serves as a simulated lecture content PDF that the lecturer or professor uses to present slides.)

```
def read_pdf(pdf_file_path):
    """Reads text from a PDF file."""
    try:
        with open(pdf_file_path, 'rb') as pdf_file:
            pdf_reader = PyPDF2.PdfReader(pdf_file)
            text = ""
            for page_num in range(len(pdf_reader.pages)):
                page = pdf_reader.pages[page_num]
                text += page.extract_text()
            return text
    except FileNotFoundError:
        print(f"Error: File not found at '{pdf_file_path}'")
        return None
    except PyPDF2.errors.PdfReadError:
        print(f"Error: Could not read PDF file '{pdf_file_path}'. It might be corrupted.")
        return None
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        return None

def extract_text_from_images_in_pdf(pdf_path):
    """Extracts text from images embedded within a PDF using OCR."""
    try:
        images = convert_from_path(pdf_path)
        ocr_text = ""
        for i, img in enumerate(images):
            print(f"Processing image from page {i+1} for OCR...")
```

```

        text = pytesseract.image_to_string(img)
        ocr_text += text + "\n---\n"
    return ocr_text
except FileNotFoundError:
    print(f"Error: File not found at '{pdf_path}'")
    return None
except Exception as e:
    print(f"An unexpected error occurred during OCR: {e}")
    return None

pdf_path = "/content/NovaTech_Q1_2025_Financial_Report.pdf"
print(f"\nReading text from PDF: {pdf_path} (direct text extraction)...")
pdf_text = read_pdf(pdf_path)

if pdf_text:
    print("\n--- PDF Direct Text Content (first 500 characters) ---")
    print(pdf_text[:500] + "..." if len(pdf_text) > 500 else pdf_text)
else:
    print("Could not extract direct text content from PDF.")

print(f"\nPerforming OCR on images in PDF: {pdf_path}...")
ocr_content = extract_text_from_images_in_pdf(pdf_path)

if ocr_content:
    print("\n--- Text Extracted from Images (OCR) ---")
    print(ocr_content[:500] + "..." if len(ocr_content) > 500 else ocr_content)
else:
    print("Could not extract text from images using OCR.")

# Combine all textual content for chunking
combined_document_text = (transcript_text if transcript_text else "") + \
    (pdf_text if pdf_text else "") + \
    (ocr_content if ocr_content else "")

```



Reading text from PDF: /content/NovaTech_Q1_2025_Financial_Report.pdf (dire

--- PDF Direct Text Content (first 500 characters) ---

NovaTech Ltd. – Q1 2025 Financial Report

1. Executive Summary

NovaTech Ltd., a mid-sized technology solutions provider, reported solid pe
increased by 12% year-over-year, driven by strong demand in cloud services
Net profit rose modestly due to increased operating expenses and R&D invest

2. Key Financial Highlights

Metric Q1 2025 Q1 2024 Change (%)

Revenue £42.3 million £37.7 million +12.2%

Gross Profit £25.6 million £22.8 million...

Performing OCR on images in PDF: /content/NovaTech_Q1_2025_Financial_Report

Processing image from page 1 for OCR...

Processing image from page 2 for OCR...

--- Text Extracted from Images (OCR) ---

NovaTech Ltd. – Q1 2025 Financial Report

1. Executive Summary

NovaTech Ltd., a mid-sized technology solutions provider, reported solid pe
increased by 12% year-over-year, driven by strong demand in cloud services

Net profit rose modestly due to increased operating expenses and R&D invest

2. Key Financial Highlights

Metric Q1 2025 Q1 2024 Change (%)

Revenue £42.3 million £37.7 million +12.2%

Gross ...

✓ Text Chunking and Embeddings

This section outlines the critical processes for preparing extracted textual content for efficient semantic search and retrieval, particularly beneficial for large datasets. This preparation involves:-

1. The first step, **text chunking**, utilizes the `semantic_chunk_document` function to segment the `combined_document_text` (comprising content from both the audio transcript and the PDF, including direct and OCR-extracted text) into smaller, manageable units called "chunks." A `RecursiveCharacterTextSplitter` is employed for this purpose, which intelligently divides text based on common delimiters such as newlines, periods, and spaces. This method ensures that individual chunks maintain semantic coherence, preventing the fragmentation of sentences or ideas.
2. A pre-trained transformer model, `all-MiniLM-L6-v2`, is loaded with the primary function to convert textual content into **embeddings** to capture the semantic meaning of the text.
3. The `get_hf_embeddings` function then leverages this model to create an embedding vector for each of the previously generated document chunks, which are fundamental for enabling **semantic search capabilities**.
4. Finally, a **FAISS (Facebook AI Similarity Search) index** is constructed, which is a specialized library engineered for efficient similarity search and clustering of dense vectors. An `IndexFlatL2` type index is instantiated, providing a simple and rapid mechanism for exact nearest neighbor searches based on Euclidean distance (L2 norm). The generated `chunk_embeddings` are subsequently added to this index. Concurrently, an `id_to_chunk_map` is established, facilitating the straightforward retrieval of the original text chunk using its corresponding index ID.

By intelligently chunking the text and converting these chunks into searchable embeddings stored within a FAISS index, this process enables the rapid and efficient retrieval of semantically relevant information, moving beyond mere keyword matching, representing a foundational component for developing advanced question-answering systems and intelligent search functionalities across extensive document corpuses.

```

def semantic_chunk_document(text, chunk_size=500, chunk_overlap=50):
    """Chunks text into smaller, semantically coherent pieces."""
    if not text:
        return []
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=chunk_size,
        chunk_overlap=chunk_overlap,
        separators=["\n\n", "\n", ".", "!", "?", " ", ""],
    )
    chunks = text_splitter.split_text(text)
    return chunks

all_document_chunks = semantic_chunk_document(combined_document_text)
print(f"\nTotal chunks created: {len(all_document_chunks)}")

print("Loading embedding model 'all-MiniLM-L6-v2'...")
hf_embedding_model = SentenceTransformer('all-MiniLM-L6-v2')

def get_hf_embeddings(texts):
    """Helper function to get embeddings using the loaded HF model."""
    return hf_embedding_model.encode(texts, convert_to_tensor=True)

print("Building FAISS index for document chunks...")
if all_document_chunks:
    chunk_embeddings = get_hf_embeddings(all_document_chunks).cpu().numpy()
    dimension = chunk_embeddings.shape[1]
    index = faiss.IndexFlatL2(dimension)
    index.add(chunk_embeddings)
    id_to_chunk_map = {i: chunk for i, chunk in enumerate(all_document_chunks)}
    print("FAISS index built successfully.")
else:
    print("No chunks available to build FAISS index.")
    index = None
    id_to_chunk_map = {}

```



```

Total chunks created: 15
Loading embedding model 'all-MiniLM-L6-v2'...
Building FAISS index for document chunks...
FAISS index built successfully.

```

✓ Retrieval (BM25 and Hybrid)

This section outlines the retrieval mechanism, detailing how the system identifies the most relevant information from the processed document chunks based on a user's query. This approach integrates both a traditional keyword-based search and a more advanced semantic search to achieve enhanced retrieval accuracy.

BM25 Index Creation (Keyword Search)

1. A **BM25 (Best Match 25) index** is constructed, which is a widely recognized and effective ranking function in information retrieval. It assesses the relevance of documents by considering the frequency of query terms within a document and their inverse document frequency across the entire collection. This enables the identification of document chunks that are highly relevant based on exact keyword matches. The `all_document_chunks` are tokenized and subsequently utilized to build this index.
2. The `hybrid_retrieval` function is designed to combine the strengths of two distinct retrieval methodologies:
 - **Vector Search (FAISS)**: This component leverages the previously constructed **FAISS index** and the **query's embedding** to locate document chunks that are semantically similar to the user's query. This is crucial for discovering relevant information even when exact keywords are not present in the query.
 - Concurrently, a **BM25 (Keyword Search)** is executed to identify chunks exhibiting strong keyword overlaps with the query.

The results from both the vector search and the BM25 search are then combined. By taking the union of the top candidate chunks from both methods, the system ensures a broader and more robust set of potentially relevant information. These combined chunks are returned as "initial candidates," ready for further processing, such as re-ranking or serving as contextual input for a large language model.

By implementing this **hybrid retrieval approach**, the system effectively leverages both the precise keyword matching capabilities of BM25 and the semantic understanding provided by vector embeddings. This synergistic combination leads to more comprehensive and accurate information retrieval for a given query.

```

print("Creating BM25 index...")
if all_document_chunks:
    tokenized_corpus = [chunk.lower().split(" ") for chunk in all_document_chunks]
    bm25 = BM25Okapi(tokenized_corpus)
    print("BM25 index created.")
else:
    print("No chunks available to create BM25 index.")
    bm25 = None

def hybrid_retrieval(query, top_k=5):
    """
    Performs a simplified hybrid retrieval combining vector search (FAISS) and
    Returns initial candidates for re-ranking.
    """
    if index is None or bm25 is None:
        print("Retrieval systems not initialized. Cannot perform hybrid retrieval")
        return []


    query_embedding = get_hf_embeddings([query]).cpu().numpy()
    D, I = index.search(query_embedding, top_k * 2)
    vector_results_indices = I[0]

    tokenized_query = query.lower().split(" ")
    bm25_scores = bm25.get_scores(tokenized_query)
    bm25_results_indices = np.argsort(bm25_scores)[::-1][:top_k * 2]

    combined_candidates_indices = set(vector_results_indices).union(set(bm25_results_indices))
    retrieved_chunks = [id_to_chunk_map[idx] for idx in combined_candidates_indices]

    return retrieved_chunks

```

 Creating BM25 index...
 BM25 index created.

✓ Re-ranking

This section introduces a critical stage for elevating the quality of search results: **re-ranking**. While the preceding retrieval mechanism identifies a broader set of potentially relevant documents through a hybrid approach of keyword and semantic search, the re-ranking process refines these initial candidates to present the most accurate and pertinent information.

The re-ranking mechanism operates as follows:

- A specialized **re-ranker model** (`BAAI/bge-reranker-base`) is loaded. Unlike embedding models, which produce general semantic representations, re-ranker models are specifically trained to evaluate and score the relevance of a given document in relation to a particular query. These models accept a query-document pair (or similar textual snippets) and output a numerical score indicative of the document's relevance to the query. The associated tokenizer and model are initialized and set to evaluation mode (`.eval()`) to optimize for inference.
- **rerank_documents Function:** This function accepts a user's `query` and a list of `retrieved_documents` (which constitute the initial candidates from the hybrid retrieval step). It systematically constructs pairs of `[query, document]` for each retrieved document. These pairs are then fed into the re-ranker model, which computes a **relevance score** for each pair. Subsequently, the documents are sorted in descending order based on these relevance scores, ensuring that the most relevant documents are positioned at the forefront of the list.
- **retrieve_relevant_passages Function:** This overarching function orchestrates the entire retrieval process. It initially invokes the `hybrid_retrieval` function (from the previous stage) to obtain an initial set of relevant document chunks (e.g., `top_k_retrieval=10` candidates). These initial chunks are then passed to the `rerank_documents` function for refinement. Finally, the function selects the `top_k_rerank` (e.g., 5) highest-scoring documents from the re-ranked list for final presentation.

By integrating this re-ranking step, the system significantly enhances the precision of the retrieved information. This ensures that the final passages presented to the user are the most pertinent to their query, even when selected from a larger pool of initial candidates.

```

print("Loading re-ranker model 'BAAI/bge-reranker-base'...")
reranker_tokenizer = AutoTokenizer.from_pretrained("BAAI/bge-reranker-base")
reranker_model = AutoModelForSequenceClassification.from_pretrained("BAAI/bge-reranker-base")
reranker_model.eval()
print("Re-ranker model loaded.")

def rerank_documents(query, retrieved_documents):
    """
    Re-ranks a list of retrieved documents based on their relevance to the query.
    """
    if not retrieved_documents:
        return []

    pairs = [[query, doc] for doc in retrieved_documents]
    inputs = reranker_tokenizer(pairs, padding=True, truncation=True, return_tensors='pt')

    with torch.no_grad():
        scores = reranker_model(**inputs).logits.squeeze(dim=1)

    ranked_indices = torch.argsort(scores, descending=True).tolist()
    reranked_docs = [retrieved_documents[i] for i in ranked_indices]
    return reranked_docs

def retrieve_relevant_passages(query, top_k_retrieval=10, top_k_rerank=5):
    """
    Performs hybrid retrieval (vector + BM25) and re-ranks the results.
    """
    initial_retrieved_chunks = hybrid_retrieval(query, top_k=top_k_retrieval)
    final_retrieved_passages = rerank_documents(query, initial_retrieved_chunks)
    return final_retrieved_passages

```

➔ Loading re-ranker model 'BAAI/bge-reranker-base'...
Re-ranker model loaded.

✓ Gemini Model Setup and RAG Pipeline Execution

This final section integrates all preceding components to demonstrate the construction and utilization of a **Retrieval-Augmented Generation (RAG)** system for question answering. It encompasses the configuration of the Gemini model, the formulation of context-aware prompts, and the execution of a comprehensive RAG pipeline with automated and qualitative evaluation.

1. The initial step involves securely loading the **Google Gemini API key** for authenticating

and establishing communication with the Gemini LLM.

2. Then the `create_gemini_prompt` function is defined to structure the input for the Gemini model, intelligently combining the user's query with the `relevant_chunks` retrieved in the preceding steps (via BM25 and hybrid re-ranking). By furnishing the Gemini model with specific contextual information, this approach actively guides the model to generate responses grounded exclusively in the provided data, thereby mitigating the risk of "hallucinations" or the provision of generalized knowledge.
3. Then the `gemini-1.5-flash` version of the Gemini model is selected for its lightweight nature and efficiency, making it well-suited for rapid response generation in a RAG system.
4. RAG Question Answering System Execution and Evaluation: The core of this section involves the execution and evaluation of the RAG system using a set of predefined queries:
 - Automated Test Queries: A collection of `test_queries_with_ground_truth` is utilized. These queries are specifically designed to simulate user interactions and are associated with "expected keywords" or "expected chunk content" for a foundational level of automated evaluation.
 - Per-Query Execution: For each test query:
 - The `retrieve_relevant_passages` function (from the re-ranking stage) is invoked to fetch the most pertinent chunks from the processed document corpus.
 - A basic recall score is calculated. This metric quantifies the proportion of "expected content items" that were successfully present within the retrieved passages, providing a quantitative assessment of the retrieval component's performance.
 - If relevant passages are identified, a prompt is dynamically constructed using both the query and the retrieved context. The Gemini model then generates an answer based on this augmented prompt.
 - The generated answer is displayed, accompanied by clear prompts for manual review. This qualitative evaluation is paramount for assessing the factual correctness, comprehensiveness, and fluency of the LLM's response.
 - Overall Summary: Upon completion of all test executions, an average retrieval

recall score is computed and presented, summarizing the overall performance of the retrieval component.

5. **Interactive RAG Session (Optional):** An optional interactive loop is implemented, additionally allowing users to input their own questions. For each user-submitted question, the complete RAG pipeline is executed: relevant passages are retrieved and displayed, followed by the generation of a Gemini response strictly based on the provided contextual passages.

In summary, this section provides a comprehensive demonstration of a fully operational RAG system. It seamlessly integrates all preceding stages—including audio and PDF processing, text chunking, embedding generation, and sophisticated retrieval—to empower a large language model like Gemini to deliver accurate and transparent answers by grounding its responses in specific, retrieved document content.

```
print("Attempting to configure Google Gemini API key...")
api_key = None
try:
    api_key = os.environ["GOOGLE_API_KEY"]
    genai.configure(api_key=api_key)
    print("API key loaded from environment variables.")
except KeyError:
    try:
        from google.colab import userdata
        api_key = userdata.get('GOOGLE_API_KEY')
        genai.configure(api_key=api_key)
        print("API key loaded from Colab secrets.")
    except Exception as e:
        print(f"Could not retrieve API key from environment or Colab secrets: {e}")
        print("Please set the GOOGLE_API_KEY environment variable or add it as a secret")

def to_markdown(text):
    text = text.replace('•', ' *')
    return Markdown(textwrap.indent(text, '> ', predicate=lambda x: True))

def create_gemini_prompt(query, relevant_chunks):
    context = "\n".join(relevant_chunks)
    prompt = f"""Based on the following context, answer the question thoroughly :
    If the context does not contain enough information to answer the question, state

    Context:
    {context}

    Question: {query}
```



```

Answer: ""
    return prompt

gemini_model = None
if api_key:
    try:
        gemini_model = genai.GenerativeModel('gemini-1.5-flash')
        print("Initialized Gemini model: gemini-1.5-flash")
    except Exception as e:
        print(f"Error initializing Gemini model: {e}")
else:
    print("Gemini model not initialized: API key is missing.")

print("\n--- Starting RAG Question Answering System ---")

# --- Automated Test Queries ---
print("\n--- Running Automated Test Queries ---")

# Define a set of test queries and their EXPECTED RELEVANT CHUNKS/KEYWORDS.
# This is crucial for quantitative retrieval evaluation.
# For a real system, you'd manually curate these "ground truth" associations.
# I'm providing conceptual examples here based on typical document content.
test_queries_with_ground_truth = {
    "What are the key financial results for NovaTech in Q1 2025?": {
        "expected_keywords": ["revenue", "profit", "expenses", "earnings", "q1 2025"],
        "expected_chunks_content": ["NovaTech Ltd. - Q1 2025 Financial Report", '
    },
    "Can you tell me about the transcription content?": {
        "expected_keywords": ["transcript", "audio", "financial performance", "NovaTech"],
        "expected_chunks_content": ["Novartek limited solid quarter one of the 2025"],
    },
    "What is the general topic of the transcribed audio?": {
        "expected_keywords": ["audio", "topic", "summary", "NovaTech"],
        "expected_chunks_content": ["NovaTech Ltd. - Q1 2025 Financial Report", '
    },
    "Is there any information about NovaTech's earnings in the PDF?": {
        "expected_keywords": ["earnings", "profit", "nova", "report"],
        "expected_chunks_content": ["Net Income", "EPS (Basic)", "Gross Profit", '
    },
    "Summarize the main points from the documents.": {
        "expected_keywords": ["summary", "key points", "overall", "NovaTech", "financial"],
        "expected_chunks_content": ["NovaTech Ltd. - Q1 2025 Financial Report", '
    },
    "What details are available from the images in the PDF?": {
        "expected_keywords": ["image", "ocr", "scanned", "table"],
        "expected_chunks_content": ["Metric", "Q1 2025", "Q1 2024", "Change (%)", '
    },
    "What are NovaTech's revenues?": {
        "expected_keywords": ["revenue", "income", "sales", "gross", "net"],
        "expected_chunks_content": ["NovaTech Ltd. - Q1 2025 Financial Report", '
    },
}

```

```

        "expected_keywords": ["revenue", "sales", "income", "q1 2025"],
        "expected_chunks_content": ["Revenue", "£42.3 million", "£37.7 million",
    },
    "How does the audio relate to the PDF content?": {
        "expected_keywords": ["relate", "connection", "link", "audio", "pdf", "No",
        "expected_chunks_content": ["NovaTech Ltd. – Q1 2025 Financial Report", '
    }
}

# Metrics to track
retrieval_recall_scores = []
generation_feedback = [] # Store manual review notes for generation

for i, query_text in enumerate(test_queries_with_ground_truth.keys()):
    expected_info = test_queries_with_ground_truth[query_text]
    print(f"\n==== Running Test Query {i+1}/{len(test_queries_with_ground_truth)}")
    print(f"Query: {query_text}")

    if gemini_model and index is not None and bm25 is not None and reranker_model:
        relevant_passages = retrieve_relevant_passages(query_text, top_k_retrieval)

        # --- Evaluate Retrieval (Quantitative – Basic Recall) ---
        retrieved_content_lower = " ".join(relevant_passages).lower()
        num_expected_found_in_retrieval = 0
        if expected_info["expected_chunks_content"]: # Check if there are expected chunks
            for ec in expected_info["expected_chunks_content"]:
                if ec.lower() in retrieved_content_lower:
                    num_expected_found_in_retrieval += 1
            # Simple Recall: (num expected found) / (total num expected)
            recall = num_expected_found_in_retrieval / len(expected_info["expected_chunks_content"])
            retrieval_recall_scores.append(recall)
            print(f" Retrieval Recall (basic): {recall:.2f} ({num_expected_found_in_retrieval}/{len(expected_info['expected_chunks_content'])})")
        else:
            # If no specific content expected, assume full recall if passages are relevant
            recall = 1.0 if relevant_passages else 0.0
            retrieval_recall_scores.append(recall)
            print(f" Retrieval Recall (basic): N/A (no specific expected content)")

    if relevant_passages:
        # print("\nRetrieved Passages for the query:") # Uncomment for details
        # for j, passage in enumerate(relevant_passages):
        #     print(f"Passage {j+1}:\n{passage[:200]}...") # Print only first 200 characters

        gemini_prompt = create_gemini_prompt(query_text, relevant_passages)

        try:
            gemini_response = gemini_model.generate_content(gemini_prompt)
            generated_answer = gemini_response.text

```

```

generated_answer = gemini_response_text
# print("\nGemini Response (Full):") # Uncomment for detailed del
# display(to_markdown(generated_answer))

# --- Evaluate Generation (Qualitative/Manual for now) ---
print("\n--- Evaluating Generation ---")
print("  Generated Answer:")
print(textwrap.indent(generated_answer, '    '))
print("\n  Manual Review Prompt:")
print("    1. Is the answer factually correct based ONLY on the pr
print("    2. Does the answer directly and comprehensively address
print("    3. Is the language fluent and coherent? (Yes/No)")
# You would ideally store these manual reviews in a list:
# generation_feedback.append({"query": query_text, "answer": gene

except Exception as e:
    print(f"Error generating Gemini response for query '{query_text}'
    # Log the error for evaluation
    generation_feedback.append({"query": query_text, "answer": "ERROR

else:
    print(f"No relevant passages found for query: '{query_text}'. Genera
    generation_feedback.append({"query": query_text, "answer": "N/A", "ma

else:
    print("RAG system components are not fully initialized. Cannot run tests
    break # Exit the test loop if components aren't ready

print("\n--- Automated Test Queries Complete ---")

# --- Summarize Metrics (if tests ran) ---
if retrieval_recall_scores:
    avg_recall = np.mean(retrieval_recall_scores)
    print(f"\n--- Overall Evaluation Summary ---")
    print(f"Average Retrieval Recall (based on expected content keywords): {avg_r
    print("\nFor detailed generation evaluation, manually review the outputs abov
    print("Consider setting up a more robust evaluation framework for production

# --- Interactive Loop (Optional for additional interaction and questioning) ---
print("\n--- Starting Interactive RAG Session (Type 'exit' to quit) ---")
while True:
    user_question = input("\nEnter your question: ")

    if user_question.lower() == 'exit':
        print("Exiting the RAG system. Goodbye!")
        break

    if gemini_model and index is not None and bm25 is not None and reranker_mode
        relevant_passages = retrieve_relevant_passages(user_question, top_k_retr

```

```

if relevant_passages:
    print("\nRetrieved Passages for the query:")
    for i, passage in enumerate(relevant_passages):
        print(f"Passage {i+1}:\n{passage}\n---")

    gemini_prompt = create_gemini_prompt(user_question, relevant_passages)

    try:
        gemini_response = gemini_model.generate_content(gemini_prompt)
        print("\nGemini Response:")
        try:
            display(to_markdown(gemini_response.text))
        except NameError:
            print(gemini_response.text)
    except Exception as e:
        print(f"Error generating Gemini response: {e}")
    else:
        print("No relevant passages found for the query.")
else:
    print("RAG system components are not fully initialized. Cannot generate")

```

```

Attempting to configure Google Gemini API key...
API key loaded from Colab secrets.
Initialized Gemini model: gemini-1.5-flash

--- Starting RAG Question Answering System ---

--- Running Automated Test Queries ---

==== Running Test Query 1/8 ====
Query: What are the key financial results for NovaTech in Q1 2025?
Retrieval Recall (basic): 0.67 (4/6 expected content items found)

--- Evaluating Generation ---
Generated Answer:
NovaTech Ltd. reported a 12.2% year-over-year revenue increase, reaching

Manual Review Prompt:
1. Is the answer factually correct based ONLY on the provided context? (Yes/No)
2. Does the answer directly and comprehensively address the query? (Yes/No)
3. Is the language fluent and coherent? (Yes/No)

==== Running Test Query 2/8 ====
Query: Can you tell me about the transcription content?
Retrieval Recall (basic): 0.86 (6/7 expected content items found)

--- Evaluating Generation ---
Generated Answer:
The transcription covers NovaTech Ltd.'s Q1 2025 financial report. Key

```

Manual Review Prompt:

1. Is the answer factually correct based ONLY on the provided context? (Y/N)
2. Does the answer directly and comprehensively address the query? (Yes/No)
3. Is the language fluent and coherent? (Yes/No)

==== Running Test Query 3/8 ====

Query: What is the general topic of the transcribed audio?

Retrieval Recall (basic): 0.67 (2/3 expected content items found)

--- Evaluating Generation ---

Generated Answer:

The general topic is NovaTech Ltd.'s Q1 2025 financial report and outlo

Manual Review Prompt:

1. Is the answer factually correct based ONLY on the provided context? (Y/N)
2. Does the answer directly and comprehensively address the query? (Yes/No)
3. Is the language fluent and coherent? (Yes/No)

==== Running Test Query 4/8 ====

Query: Is there any information about NovaTech's earnings in the PDF?

Retrieval Recall (basic): 1.00 (4/4 expected content items found)

--- Evaluating Generation ---

Generated Answer:

Yes. NovaTech's Q1 2025 financial report shows a Net Income of £6.1 mi

Manual Review Prompt:

1. Is the answer factually correct based ONLY on the provided context? (Y/N)
2. Does the answer directly and comprehensively address the query? (Yes/No)
3. Is the language fluent and coherent? (Yes/No)

==== Running Test Query 5/8 ====

Query: Summarize the main points from the documents.

Retrieval Recall (basic): 0.82 (9/11 expected content items found)

--- Evaluating Generation ---

Generated Answer:

NovaTech Ltd. had a strong Q1 2025, with a 12% YoY revenue increase to

Manual Review Prompt:

1. Is the answer factually correct based ONLY on the provided context? (Y/N)
2. Does the answer directly and comprehensively address the query? (Yes/No)
3. Is the language fluent and coherent? (Yes/No)

==== Running Test Query 6/8 ====

Query: What details are available from the images in the PDF?

Retrieval Recall (basic): 0.47 (9/19 expected content items found)

--- Evaluating Generation ---

Generated Answer:

The provided text is a financial report for NovaTech Ltd. for Q1 2025.

Manual Review Prompt:

1. Is the answer factually correct based ONLY on the provided context? (Y/N)
2. Does the answer directly and comprehensively address the query? (Yes/No)
3. Is the language fluent and coherent? (Yes/No)

==== Running Test Query 7/8 ====

Query: What are NovaTech's revenues?

Retrieval Recall (basic): 0.57 (4/7 expected content items found)

--- Evaluating Generation ---

Generated Answer:

NovaTech Ltd.'s total revenue for Q1 2025 was £42.3 million.

Manual Review Prompt:

1. Is the answer factually correct based ONLY on the provided context? (Y/N)
2. Does the answer directly and comprehensively address the query? (Yes/No)
3. Is the language fluent and coherent? (Yes/No)

==== Running Test Query 8/8 ====

Query: How does the audio relate to the PDF content?

Retrieval Recall (basic): 0.44 (4/9 expected content items found)

--- Evaluating Generation ---

Generated Answer:

There is no audio mentioned in the provided text. Therefore, I cannot

Manual Review Prompt:

1. Is the answer factually correct based ONLY on the provided context? (Y/N)
2. Does the answer directly and comprehensively address the query? (Yes/No)
3. Is the language fluent and coherent? (Yes/No)

--- Automated Test Queries Complete ---

--- Overall Evaluation Summary ---

Average Retrieval Recall (based on expected content keywords): 0.69

For detailed generation evaluation, manually review the outputs above. Consider setting up a more robust evaluation framework for production.

--- Starting Interactive RAG Session (Type 'exit' to quit) ---

Enter your question: Are the aforementioned answers factually correct based

Retrieved Passages for the query:

Passage 1:

. Enterprise software grew by 9.8% to 8.3 million pounds benefiting from de

Passage 2:

- Expected increase in marketing spend by 8% to support European expansionN

Passage 3:

Gross margin: 62%

4. Balance Sheet Snapshot (as of March 31, 2025)

Asset/Liability Amount (£ millions)

Total Assets 112.4

Total Liabilities 48.7

Shareholder's Equity 63.7

Current Ratio 2.4

Debt-to-Equity Ratio 0.45

5. Strategic Developments

- Launched NovaCloud 2.0 with improved scalability and security features.
- Acquired a small AI startup in Cambridge for £2.3 million.

Passage 4:

.1 million pounds this reflects our strategic investments in operating expe

Passage 5:

. We are now open to your questions.NovaTech Ltd. - Q1 2025 Financial Report

Passage 6:

- Signed a strategic partnership with a German logistics company to develop

6. Outlook for Q2 2025

- Projected revenue: £45-47 million
- Continued focus on AI integration and SME cloud adoption

```

- Expected increase in marketing spend by 8% to support European expansion
[]
---
---
Passage 7:
.0 the acquisition of a ai startup in cambridge and a new partnership with
---

Gemini Response:

    Yes

Enter your question: exit
Exiting the RAG system. Goodbye!

```

✓ Video Generation (Optional)

This optional, final component demonstrates a novel approach to visualize the document processing and simulated learning progress through video generation, leveraging the previously processed text chunks. Depending on the user's selection, the system can generate either a simple text-based video or a more illustrative visual video.

```

choice = "Generate Visual Video" #@param

if not all_document_chunks:
    print("No document chunks available")
else:
    if choice == "Generate Text Video":
        print("\n--- Generating Text Video")
        combined_segments = all_document_chunks
        fig, ax = plt.subplots(figsize=(10, 10))
        ax.set_xlim(0, 10)
        ax.set_ylim(0, 10)
        ax.axis('off')
        text_display = ax.text(5, 5, "")

    def update_text_video(frame):
        if frame < len(combined_segments):
            segment = combined_segments[frame]
            wrapped_text = "\n".join(segment.split("\n"))
            text_display.set_text(wrapped_text)
        else:
            text_display.set_text('')

```

choice:

Generate Visual Video




```

        return text_display,

    ani = animation.FuncAnimation(fig, update_visual_video, frames=len(combined_segments),
    try:
        print("\nSaving animated text as video")
        ani.save('animated_text_correlation.mp4')
        print("Animation saved as a video")
    except Exception as e:
        print(f"Error saving animation: {e}")
        print("Saving video requires ffmpeg installed")

elif choice == "Generate Visual Video":
    print("\n--- Generating Visual Video ---")
    combined_segments = all_document_embeddings
    fig, (ax1, ax2) = plt.subplots(2, 1)

    ax1.set_xlim(0, 10)
    ax1.set_ylim(0, 10)
    ax1.axis('off')
    text_display = ax1.text(5, 5, 'Learning Progress')

    ax2.set_xlabel("Time Step (Arbitrary)")
    ax2.set_ylabel("Value (Arbitrary)")
    ax2.set_title("Learning Progress")
    ax2.set_xlim(0, len(combined_segments))
    ax2.set_ylim(0, 1.1)
    line, = ax2.plot([], [], 'r-',
                    xdata=[], ydata=[])

    def update_visual_video(frame):
        if frame < len(combined_segments):
            segment = combined_segments[frame]
            wrapped_text = "\n".join(segment)
            text_display.set_text(wrapped_text)
        else:
            text_display.set_text('')

        if frame < len(combined_segments):
            xdata.append(frame)
            simulated_progress = 1 - (frame / len(combined_segments))
            ydata.append(simulated_progress)
            line.set_data(xdata, ydata)
            ax2.set_xlim(0, max(1, len(xdata)))
            ax2.set_ylim(0, max(1.1, max(ydata)))
        return text_display, line

    ani = animation.FuncAnimation(fig, update_visual_video, frames=len(combined_segments) + 1,
    try:

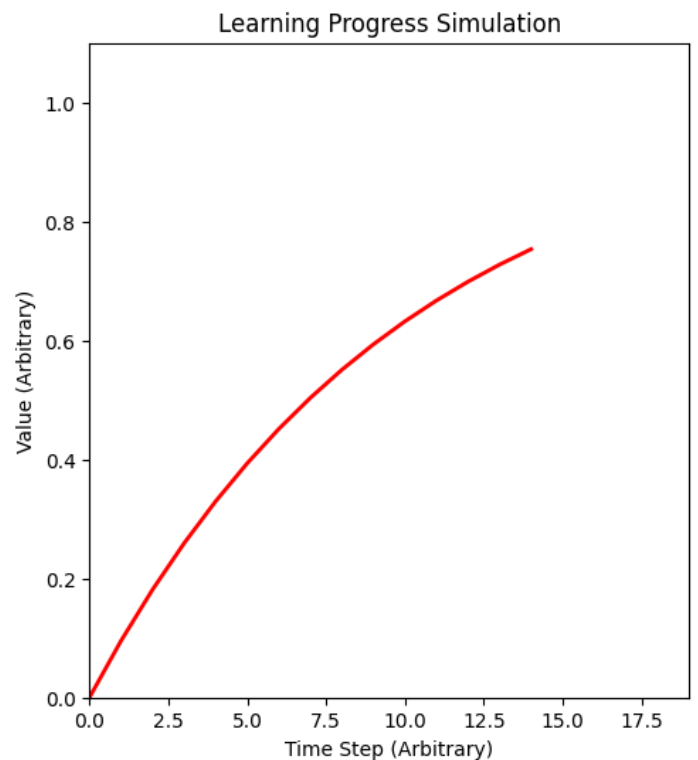
```

```
print("\nSaving animated v\nani.save('learning_animatic\nprint("Animation saved as 1\nexcept Exception as e:\nprint(f"Error saving animat\nprint("Saving video require\nelse:\nprint("Invalid choice for video
```



--- Generating Visual Video ---

Saving animated visual video. This may take a while...
Animation saved as learning_animation.mp4



✓

Table: Summary of Failure Cases and Proposed Enhancements

Failure Case	Desc
1. Loss of Semantic Structure in Tabular Data	Optical Character Recognition (OCR) flattens tables, resulting in th
2. Inability to Perform Cross-Source Reasoning	The system fails to synthesize information across heterogeneous
3. Underperformance with Abstract or Vague Queries	Abstract or underspecified queries result in suboptimal retrieval a
4. Embedding Representations Lack Domain Semantics	Off-the-shelf embeddings often miss nuanced or domain-specific