BGSW FIT.Fest GenAI Hackathon

Team Details:

Team Name: Pheonix

Name	Email	Phone No
Pranav S Bhat	01fe21bcs230@kletech.ac.in	8861668850
Ravishankar B	01fe21bcs177@kletech.ac.in	6363663113
Kushalgouda S P	kspatil.ksp@gmail.com	8105784976

Unique features implemented by us:

- **Unique features:**

- Image captioning when relevant images are fetched.
- Once an image is uploaded and sent to MML, a record is maintained of the local path to the uploaded URL.
- If the user is asking questions related to the same topic, the URL is fetched from the record and processing from model is faster
- Maintaining image history in Streamlit chat.
- Good history retaining capability

- Average PDF processing time for 2 PDFs in approach 1 (uploaded Nexon and Punch manual):
  - **1 minute 50 seconds.**
- Average Answer retrieval latency: **15 seconds (with images)**
- Average Image search latency: **7 seconds (with summary)**

- An **intelligent function** in Approach 1, to save processing time and decrease image upload latency.
- This function is disconnected from the main function, i.e: forked process which uploads all images from the PDF's to Google Drive in the background and maintains the uploaded URI as a pickle record.
- This helps when fetching relevant images while the answer is conversing with the application, the model can directly fetch the uploaded URI and need not upload it again if it has already been done.
- If the image was not yet uploaded in the backend, the normal upload takes place.

Phase 1

All KPI's achieved

1) **Method**
   - Extract all text, tables and images.
   - Create chunks, store in Vector DB.
   - When user asks question, check for presence or absence of Information.
   - Based on this ask probing questions.

Phase 2

All KPI's achieved ✓

To do the above task we have employed 2 APPROACHES:

1) **Approach 1**
    - **Our own quicker method**
    - Does not calculate image embeddings, hence faster PDF Processing
    - Finds most relevant images for the given answer based on similarity search from the page where answer is taken from

2) **Approach 2**
    - **Conventional but Slower method**
    - Calculates image descriptions and embeddings for all images when processing PDF
    - Output based on description of images, not according to page numbers

3) All images have to be compulsory uploaded to the drive for the model to work. Intelligent functions written to maintain a record to decrease time taken to upload, by reusing URI's provided by Drive after upload.

Explained in detail in further pages

Phase 2

Approach 1 - FLOW

- Parse PDF, save text and tabular embeddings in local FAISS DB. Changed from Pinecone to FAISS local as it saves time and does not need to fetch from cloud.
- Extract all images and store in suitable folders, based on page numbers.
- DOES NOT calculate image description and embeddings in Approach 1

- User asks a question, similarity search run on the Vector DB for relevant documents/chunks.
- Get all relevant chunks along with metadata like "page_number", "document_name" from both Text DB and Tabular DB.

- Store unique page numbers of relevant chunks.
- Generate answer or probing questions based on asked question.
- Get all images from the relevant pages.

- Send answer and relevant images to another Multimodal Model, to get only the most appropriate images.

- Display them to the user.

Phase 2

Approach 2 - FLOW

- Parse PDF, save text and tabular embeddings in local FAISS DB. Changed from Pinecone to FAISS local as it saves time and does not need to fetch from cloud.
- Extract all images and store in suitable folders, based on page numbers.
- Pass all images to the Multimodal Model and get description of all images
- Embed these descriptions and save in FAISS DB.
- Slow processing compared to Approach 1

- User asks a question, similarity search run on the Vector DB for relevant documents/chunks.
- Get all relevant chunks along with metadata like "page_number", "document_name", "image_path" from all Text DB, Tabular DB and Image DB.

- Generate answer and relevant images after passing context and image descriptions, to get only the most appropriate images and answer.

- Display them to the user.

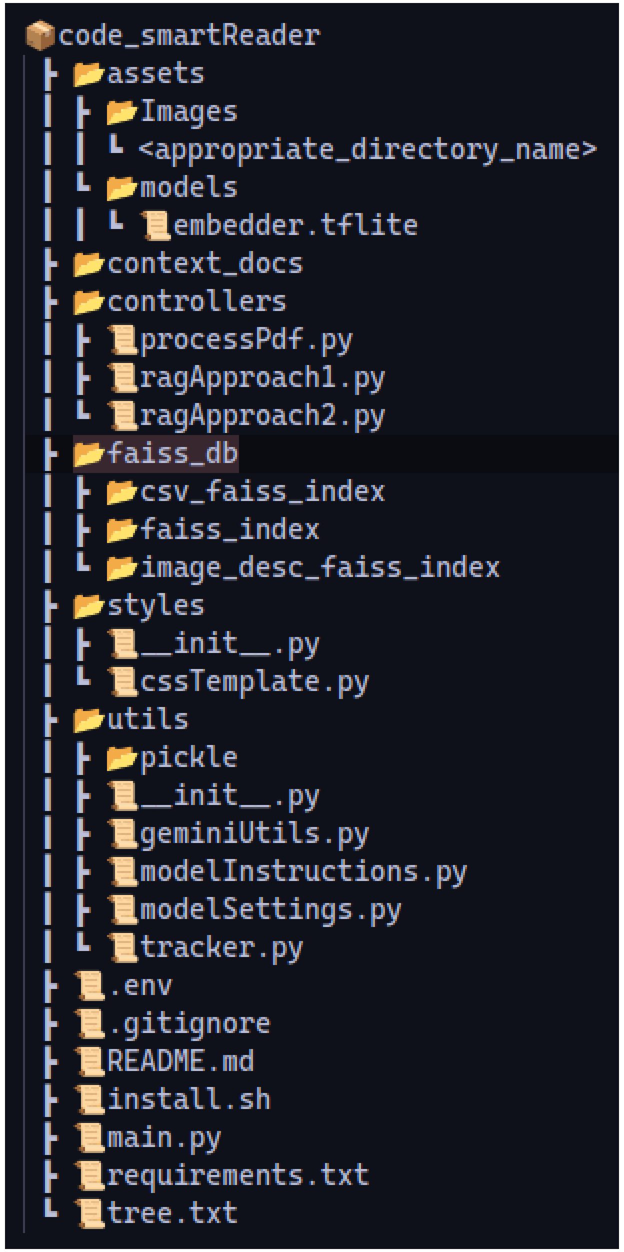
- One downside with this approach is that as the images are selected based on description of the image and not based on distance from the answer (i.e: Not necessarily on nearest pages), sometimes the images maybe wrong.

Phase 3

All KPI's achieved ✓

1) **Method**
   - Calculate all embeddings of images with the MediaPipe model.
   - Embeddings based on features of an image.
   - When user uploads any image, run cosine_similarity with the embeddings. Get the matched image.
   - We also get from which page the image belongs to.
   - Get all text from that page and surrounding pages.
   - Pass to LLM model the image, and context to get the detailed summary.

Folder Structure



```
📦code_smartReader
 ├ 📂assets
 │ ├ 📂Images
 │ │ └ <appropriate_directory_name>
 │ └ 📂models
 │ │ └ 📜embedder.tflite
 ├ 📂context_docs
 ├ 📂controllers
 │ ├ 📜processPdf.py
 │ ├ 📜ragApproach1.py
 │ └ 📜ragApproach2.py
 ├ 📂faiss_db
 │ ├ 📂csv_faiss_index
 │ ├ 📂faiss_index
 │ └ 📂image_desc_faiss_index
 ├ 📂styles
 │ ├ 📜__init__.py
 │ └ 📜cssTemplate.py
 ├ 📂utils
 │ ├ 📂pickle
 │ ├ 📜__init__.py
 │ ├ 📜geminiUtils.py
 │ ├ 📜modelInstructions.py
 │ ├ 📜modelSettings.py
 │ └ 📜tracker.py
 ├ 📜.env
 ├ 📜.gitignore
 ├ 📜README.md
 ├ 📜install.sh
 ├ 📜main.py
 ├ 📜requirements.txt
 └ 📜tree.txt
```

- **Assets**: stores extracted images and embedder model
  - Images stored in appropriate folders by pdf name along with page number

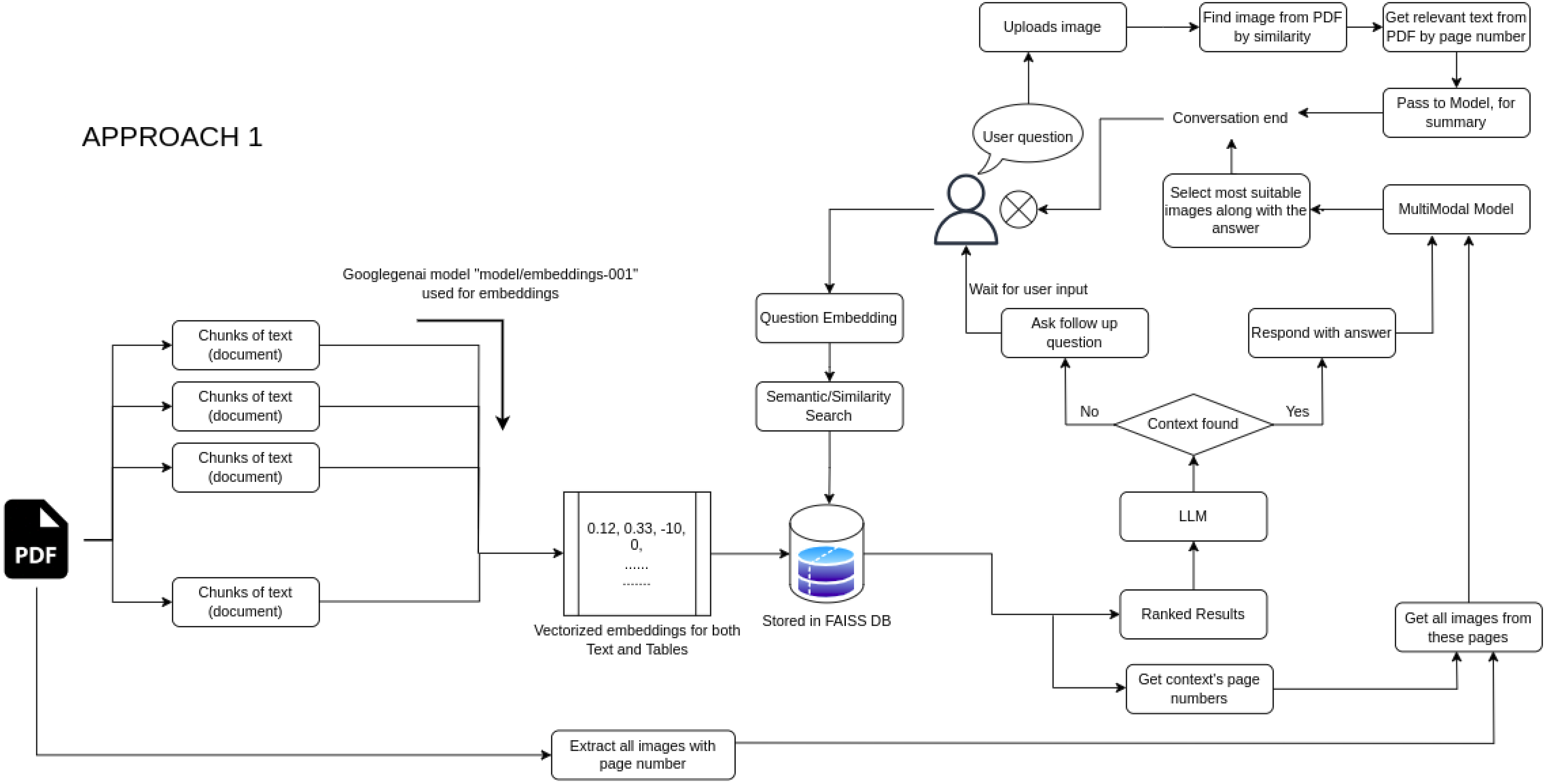
- **Context_docs**: stores all extracted table csv's, embedded image csv, etc.

- **Controllers**: The main logic of the application
  - **ProcessPDF.py**: handles the complete extraction and chunking of the PDF, ranging from text, tables and images.
  - **RagApproach1**: All model related functions, handles communication with the model based on Approach 1.
  - **RagApproach2**: Same as above but for Approach 2.

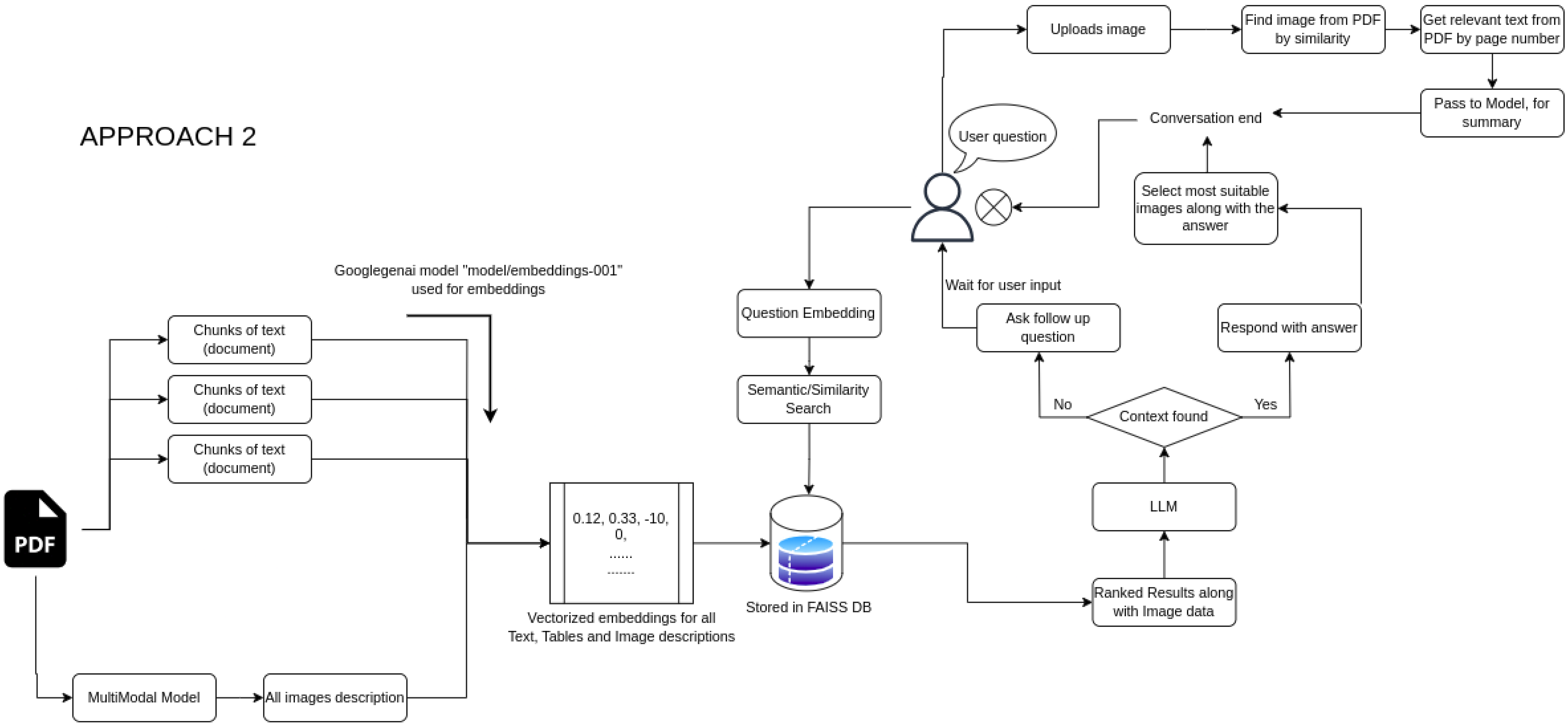
- **Utils**: All helper functions.
  - **GeminiUtils**: All uploading to drive functions
  - **ModelInstructions**: All system instructions used for LLM models.
  - **ModelSettings**: All settings used for LLM models
  - **Tracker**: To store record of image path and uploaded URI of drive for quicker retrieval.

- **Faiss_DB**: where all vector files are stored
- **.env**: Environment variables

Tech Architecture (including approach 1)


APPROACH 1
Googlegenai model "model/embeddings-001" used for embeddings
Chunks of text (document)
Chunks of text (document)
Chunks of text (document)
Chunks of text (document)
PDF
0.12, 0.33, -10, 0, ... -------
Vectorized embeddings for both Text and Tables
Stored in FAISS DB
Extract all images with page number
Question Embedding
Semantic/Similarity Search
User question
Uploads image
Find image from PDF by similarity
Get relevant text from PDF by page number
Pass to Model, for summary
Conversation end
Select most suitable images along with the answer
MultiModal Model
Wait for user input
Ask follow up question
Respond with answer
No
Context found
Yes
LLM
Ranked Results
Get context's page numbers
Get all images from these pages

Tech Architecture (including approach 2)


APPROACH 2
Googlegenai model "model/embeddings-001" used for embeddings
Chunks of text (document)
Chunks of text (document)
Chunks of text (document)
PDF
MultiModal Model
All images description
0.12, 0.33, -10, 0, ... -------
Vectorized embeddings for all Text, Tables and Image descriptions
Stored in FAISS DB
Question Embedding
Semantic/Similarity Search
User question
Uploads image
Find image from PDF by similarity
Get relevant text from PDF by page number
Pass to Model, for summary
Conversation end
Select most suitable images along with the answer
Wait for user input
Ask follow up question
Respond with answer
No
Context found
Yes
LLM
Ranked Results along with Image data

For entire DEMO and complete explanation please go through the provided video

For code documentation, comments can be found on top of functions explaining them.

IMPORTANT NOTE: In the video, at timestamp 12:48, we have misspoken Phase 3 as Approach 3.
It should have been Phase 3.

Thank You!