

Multimodal Retrieval Augmented Generation (RAG) Pipeline with ColPali and Local Visual Language Model LLaVa

1. Introduction

Retrieval Augmented Generation (RAG) is a method that combines information retrieval with generative models to provide grounded, accurate answers. This project takes RAG one step further by making it multimodal, incorporating both text and image embeddings from documents.

In this project, I utilized **ColPali** for embedding both textual and visual data, **Vespa** to store multivector embeddings as a multivector retrieval engine **BLIP** for image captioning and **LLaVA** running locally via LocalAI for multimodal generation.

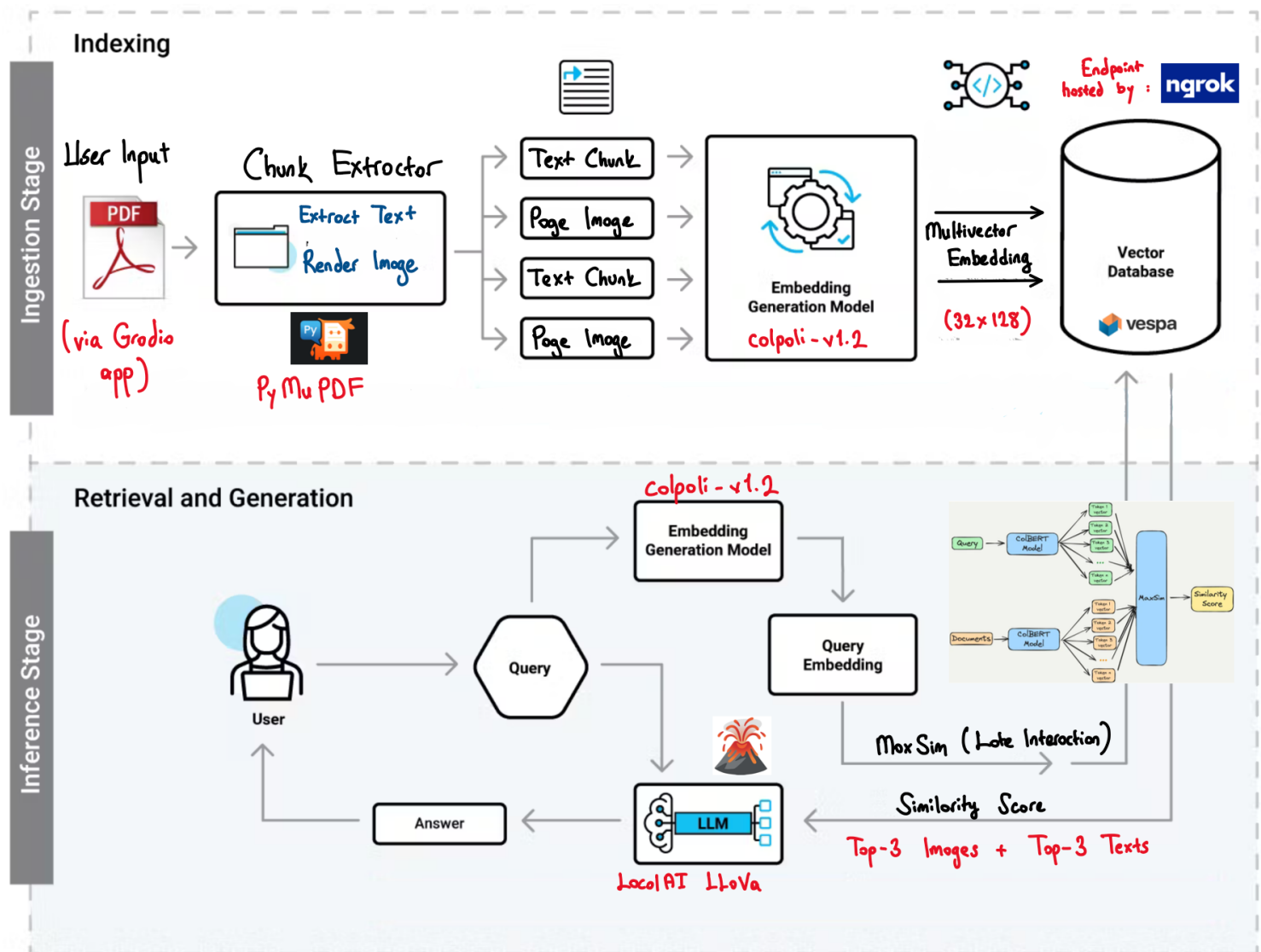
The result is a RAG pipeline that allows users to upload PDFs, retrieve the most relevant text and image data, and generate context-aware answers from a local VLM.

2. Architecture Overview

The steps of the RAG pipeline can be described as following:

1. Preprocessing PDFs: Extracting text and rendering pages to images.
2. Multimodal Embedding: Using ColPali to embed both text and image content into a shared latent space.
3. Multivector Storage: Pushing the multivector embeddings to a Vespa-powered vector database using a custom schema.
4. Retrieval: Querying the database using max-similarity scoring to fetch top text and image results.
5. Multimodal Answer Generation: Feeding image content to a locally running LLaVA model for contextual answer generation.

This architecture combines text and image understanding to fully capture the full of documents. As ColPali stands out by producing aligned embeddings for both modalities using a single lightweight model, its efficiency and tight vision-language integration make it ideal for multimodal retrieval tasks. Vespa stores these embeddings for fast querying, while MaxSim scoring ensures relevance. LLaVA then generates grounded answers from the top images—keeping everything local, fast, and privacy-preserving. The pipeline can be simply be described with the diagram below:



3. Core Components

3.1 ColPali for Multimodal Embedding

ColPali (Contrastive Pretraining of Language and Images) is a lightweight visual-language model designed for document-level understanding. It generates embeddings for both:

- **Text chunks** (extracted from PDF using PyMuPDF).
- **Rendered page images** (captured at 150 DPI, converted to PIL images).

Using ColPaliProcessor and ColPali.from_pretrained(), we obtain semantically rich embeddings from both modalities.

```
model = ColPali.from_pretrained("vidore/colpali-v1.2", ...)
processor = ColPaliProcessor.from_pretrained(...)
```

These embeddings are normalized and padded to fixed-length multivectors to align with Vespa's schema.

3.2 Image Captioning with BLIP

To provide semantic context for each image, we apply BLIP (Bootstrapping Language-Image Pretraining) to generate captions per page image. This improves searchability and summary metadata.

```
caption = blip_processor.decode(blip_model.generate(**inputs)[0],
                               skip_special_tokens=True)
```

While the final generation does not rely on captions (only raw image content), they enhance indexing metadata and traceability.

3.3 Vespa Integration for Vector Storage

Vespa is used to store and serve multimodal embeddings. The schema is designed to handle multivector tensors:

```
field embedding type tensor<float>(x[32], y[128]) {
  indexing: attribute
}
```

Documents are pushed using REST API calls, separating modality as either "text" or "image".

```
{
  "put": "id:multimodal-content:multimodal_doc::img_5",
  "fields": {
    "title": "Image page 5",
    "description": "Diagram showing missile telemetry...",
    "page": 5,
    "modality": "image",
    "embedding": { "values": [...] }
  }
}
```

3.4 Query and Retrieval via MaxSim

We encode user queries using the same ColPali processor and apply a **MaxSim** similarity scoring across all stored embeddings:

```
score = np.sum(np.max(np.matmul(query_emb, doc_emb.T), axis=1))
```

Top k results from both modalities are extracted and saved into a JSON structure (retrieved_context.json) for further processing.

4. Local Multimodal Generation with LLaVA

To stay offline and ensure data control, the generation step is delegated to **LLaVA**, running locally via **LocalAI**. No captions are used — the image content is sent directly:

```
payload = {
    "model": "llava",
    "prompt": "Answer the following question based on the visual
content of the images: ...",
    "images": [... base64 encoded images ...]
}
```

This allows LLaVA to produce **visual-grounded answers**, leveraging its fine-tuned capabilities on image-text alignment.

5. Gradio Frontend

A Gradio app provides a user-friendly interface for:

- Uploading PDFs
- Entering queries
- Viewing retrieved text passages
- Viewing top relevant pages as images

```
with gr.Blocks() as demo:
    pdf_input = gr.File(...)
    query_input = gr.Textbox(...)
    retrieve_btn = gr.Button("Retrieve")
    ...
```

6. Final Output Example

After retrieving top results and running the LLaVA generation:

- `retrieved_context.json` contains top-3 text chunks and image page numbers with captions.
- Images `page_{number}.png` are saved locally for LLaVA input.
- LLaVA returns an answer like:

```
[Response - Text #1]:  
There are 19,876 triplets in the second dataset.
```

7. Strengths of This Architecture

- **Truly Multimodal:** Retrieval considers **both text and image** modalities jointly.
- **Local Privacy:** No external APIs used — LLaVA runs offline.
- **Extendable:** The Vespa schema supports scale, and LocalAI can switch to Whisper, Bark, etc.
- **Efficient Retrieval:** Embeddings are dense, normalized, and compact.

8. Walkthrough of the Multimodal RAG Pipeline

Launching the Vespa Container

Starting a Vespa container with the following command:

```
ahmetfurkankizil@192 vespa-multimodal % docker run -d --name vespa \
-p 8080:8080 \
-p 19071:19071 \
vespaengine/vespa

07e43d8d4c6c784b663f5263f76e33f7f643517338dc2642677f5421d415e6c2
ahmetfurkankizil@192 vespa-multimodal % docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
07e43d8d4c6c   vespaengine/vespa  "/usr/local/bin/star..."  7 seconds ago  Up 7 seconds  0.0.0.0:8080->8080/tcp, 0.0.0.0:19071->19071/tcp  vespa
```

Confirming API Availability

Vespa provides a config/status API on port 19071. Verifying this with:

```
ahmetfurkankizil@192 vespa-multimodal % curl http://localhost:19071
{
  "handlers" : [ {
    "id" : "com.yahoo.document.restapi.resource.DummyDocumentV1ApiHandler",
    "class" : "com.yahoo.document.restapi.resource.DummyDocumentV1ApiHandler",
    "bundle" : "vespaclient-container-plugin:8.533.16",
    "serverBindings" : [ "http://*/document/v1/*", "http://*/document/v1/*/" ]
  }, {
    "id" : "com.yahoo.vespa.config.server.http.v2.HttpGetConfigHandler",
    "class" : "com.yahoo.vespa.config.server.http.v2.HttpGetConfigHandler",
    "bundle" : "configserver:8.533.16",
    "serverBindings" : [ "http://*/config/v2/tenant/*/application/*/*", "http://*/config/v2/tenant/*/application/*/*/*"
environment/*/region/*/instance/*/*", "http://*/config/v2/tenant/*/application/*/environment/*/region/*/instance/*/*/*" ]
  }, {
    "id" : "com.yahoo.container.usability.BindingsOverviewHandler",
    "class" : "com.yahoo.container.usability.BindingsOverviewHandler",
    "bundle" : "container-disc:8.533.16",
    "serverBindings" : [ "http://*/" ]
  }, {

```

Deploying the Application Schema

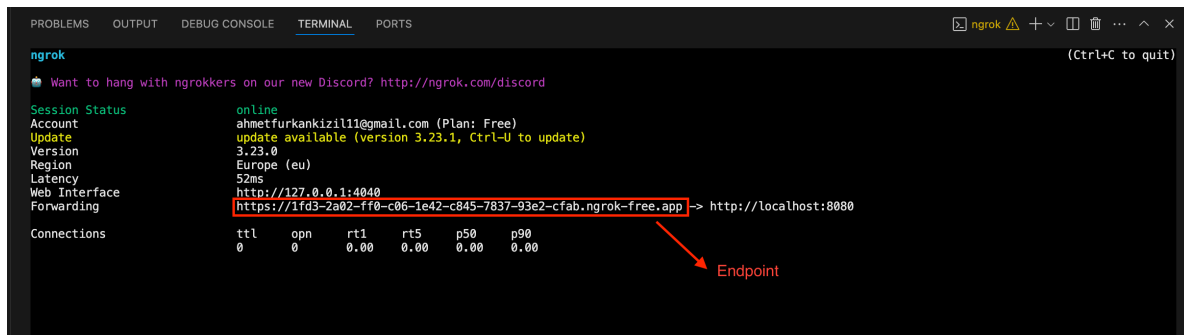
After implementing the services.xml, schema.sd, and other Vespa application files in the current directory, deploying them using:

```
ahmetfurkankizil@192 vespa-multimodal % vespa deploy
Uploading application package... done
Success: Deployed '.' with session ID 2
ahmetfurkankizil@192 vespa-multimodal %
```

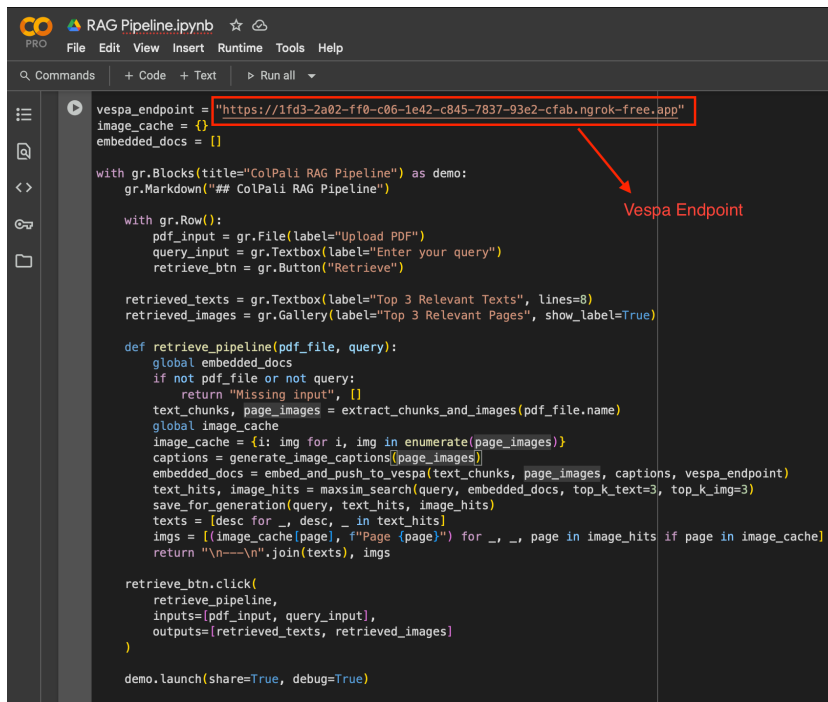
Exposing Vespa to the Web Using Ngrok

To interact with the local Vespa container from Colab or any external script, exposing it using Ngrok:

```
ngrok http 8080
```



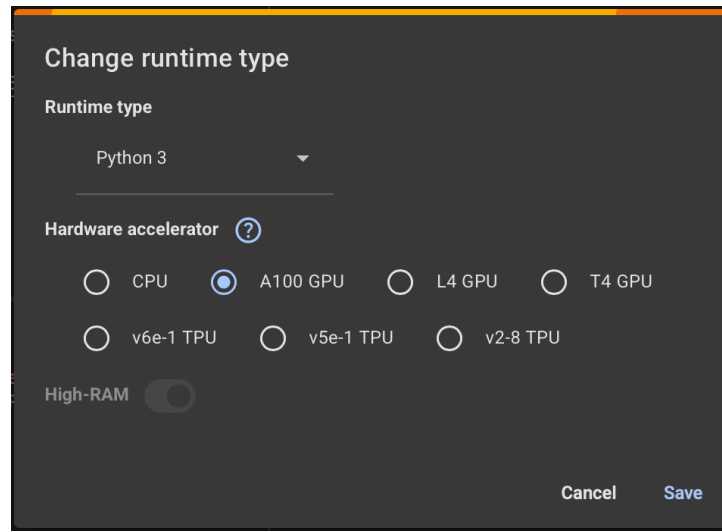
The red box shows the vespa endpoint which will be used in the Google Colab code for the storing of multivector embeddings.



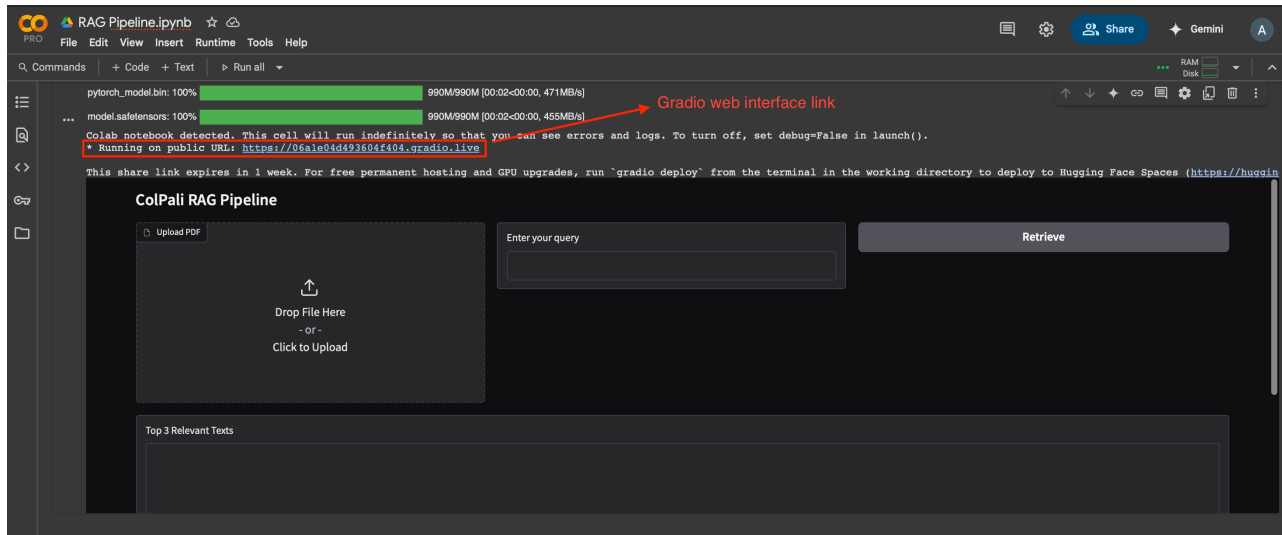
Running the Google Collab Cell to Initialize Gradio App

Since embedding models like ColPali and BLIP require substantial memory (especially during batch inference), it's important to:

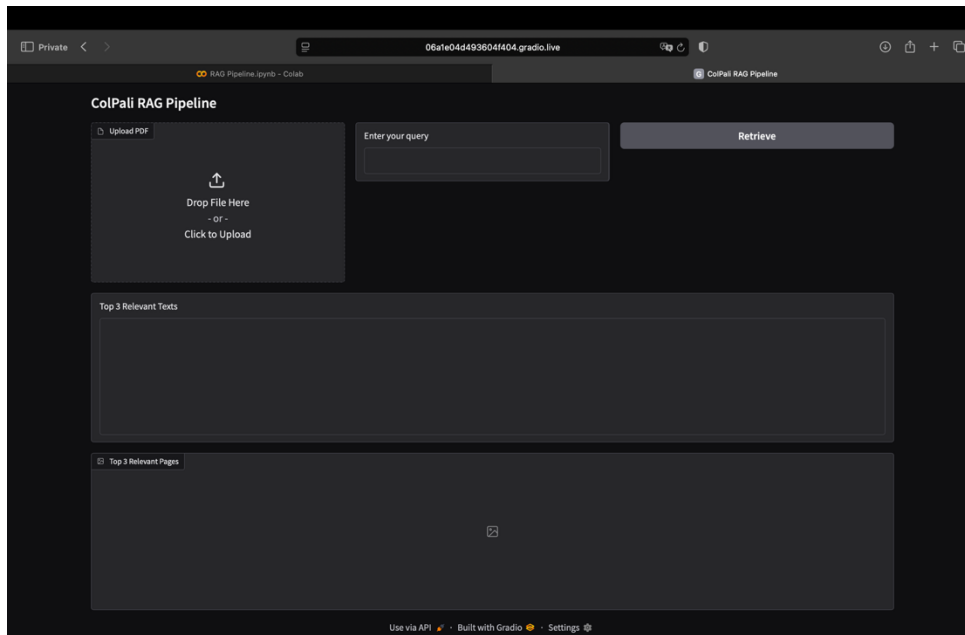
- Navigate to Runtime > Change Runtime Type.
- Setting the Hardware accelerator to A100 GPU (*recommended*).
- Optional: Toggling High-RAM if the session allows it.



Once the Runtime Type is selected, executing the Colab cell will launch a live web interface inside Colab. It will provide a public share link which is hosted by Gradio:



Clicking the Gradio link will open the web interface:



Uploading a sample pdf with entering a query starts the RAG Pipeline. The multivector embeddings generated by the ColPali model is converted into JSON format and pushed (text and images) to the Vespa database with 200 (OK) HTTP status:

```
[INFO] Embedding and pushing 42 text chunks and 8 images to Vespa...
Text doc 0 push: 200
Text doc 1 push: 200
Text doc 2 push: 200
Text doc 3 push: 200
Text doc 4 push: 200
Text doc 5 push: 200
Text doc 6 push: 200
Text doc 7 push: 200
Text doc 8 push: 200
Text doc 9 push: 200
Text doc 10 push: 200
Text doc 11 push: 200
Text doc 12 push: 200
```

...

```
Text doc 37 push: 200
Text doc 38 push: 200
Text doc 39 push: 200
Text doc 40 push: 200
Text doc 41 push: 200
Image doc img_0 push: 200
Image doc img_1 push: 200
Image doc img_2 push: 200
Image doc img_3 push: 200
Image doc img_4 push: 200
Image doc img_5 push: 200
Image doc img_6 push: 200
Image doc img_7 push: 200
[INFO] Saved retrieved_context.json and corresponding images as page_<number>.png
```


The HTTP requests are visible on the ngrok terminal as seen below:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ngrok
ngrok is also now your Kubernetes-native ingress: https://ngrok.com/r/k8s

Session Status      online
Account             ahmetfurkankizil1@gmail.com (Plan: Free)
Update              update available (version 3.23.1, Ctrl-U to update)
Version             3.23.0
Region              Europe (eu)
Latency             57ms
Web Interface        http://127.0.0.1:4040
Forwarding           https://1fd3-2a02-ff0-c06-1e42-c845-7837-93e2-cfab.ngrok-free.app -> http://localhost:8080

Connections
ttl      opn      rt1      rt5      p50      p90
132      1         0.00     0.05     90.42    90.57

HTTP Requests
15:28:50.357 +03 GET /document/v1/multimodal-content/multimodal_doc/docid/4 200 OK
15:19:34.341 +03 POST /document/v1/multimodal-content/multimodal_doc/docid/img_7 200 OK
15:19:33.143 +03 POST /document/v1/multimodal-content/multimodal_doc/docid/img_5 200 OK
15:19:33.779 +03 POST /document/v1/multimodal-content/multimodal_doc/docid/img_6 200 OK
15:19:32.886 +03 POST /document/v1/multimodal-content/multimodal_doc/docid/img_3 200 OK
15:19:32.548 +03 POST /document/v1/multimodal-content/multimodal_doc/docid/img_4 200 OK
15:19:31.375 +03 POST /document/v1/multimodal-content/multimodal_doc/docid/img_2 200 OK
15:19:30.119 +03 POST /document/v1/multimodal-content/multimodal_doc/docid/img_0 200 OK
15:19:30.861 +03 POST /document/v1/multimodal-content/multimodal_doc/docid/img_1 200 OK
15:19:28.145 +03 POST /document/v1/multimodal-content/multimodal_doc/docid/40 200 OK

```

Multivector embeddings can be seen on the ngrok endpoint with an API call:

```
GET VESPA_ENDPOINT/document/v1/multimodal-content/multimodal_doc/docid/4
```

RAG Pipeline.ipynb - Colab

ColPali RAG Pipeline

[https://f1d3-2a02-f10-c06-te42-c845-7837-93e2-clab.ngrok-free.app/document/v1/multimodal-content/multimodal_doc/docid/c41](#)

`{ "path": "/document/v1/multimodal-content/multimodal_doc/docid/c41", "id": "multimodal-content:multimodal_doc:c41", "fields": { "description": "c'm in dense vectors and that they can be used in classifying movie reviews or retrieving web pages. Undespite their success stories, little is known about how well the model works compared to Bag-of-Words/NL \"LD\" type.\"Text chunk page 0\", \"modality\": \"text\", \"page\": 0, \"embedding\": { \"type\": \"tensor(float([x(32)], y[128])), \"values\": [0.1566836063167572, 0.0265365544781532, 0.1208485366883838, ...] } }`

Once the query is submitted:

- The system embeds the query using ColPali.
- It compares this embedding to both text chunks and image captions in Vespa.
- Similarity is computed using MaxSim, returning the top results from both modalities.

Top 3 Relevant Texts: Shown in a text area. These are excerpts most relevant to the query.

Top 3 Relevant Pages: Shown as images in a gallery. These pages contain visuals or diagrams related to the query.

ColPali RAG Pipeline

Upload PDF
sample.pdf 449.2 KB

Enter your query
How many triplets in second dataset?

Retrieve

Top 3 Relevant Texts

taset consists of 19,876 triplets in which two articles are closer because they are listed in the same category by Wikipedia, and the last article is not in the same category, but may be in a sibling

To quantitatively compare these methods, we constructed two datasets for triplet evaluation. The first consists of 172 triplets of articles we knew were related because of our domain knowledge. Some

or

subject, the unrelated paper was chosen at random from papers with no shared subjects with the first paper. We produced a dataset of 20,000 triplets by this method. From the results in Table 8, it can

Top 3 Relevant Pages

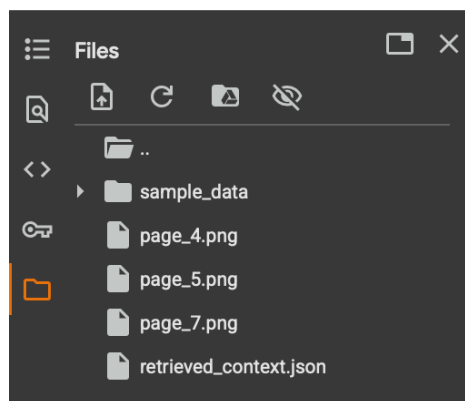
Figure 5: Results of experiments on the generated Wikipedia triplet dataset.

Table 4: Performances of different methods on dataset with generated Wikipedia triplets on the best performing dimensionality.

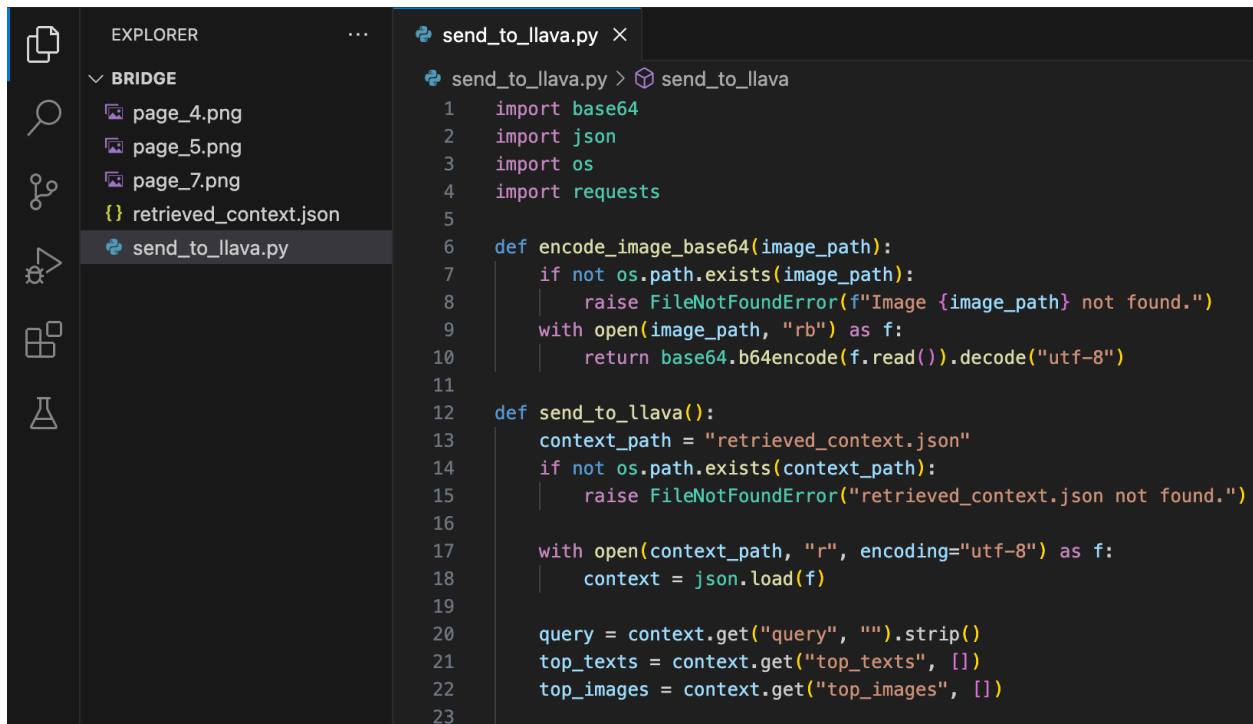
Model	Embedding dimension/triplets	Accuracy
Paragraph vectors	10000	78.8%
LDA	5000	67.7%
Averaged word embeddings	3000	74%
Bag of words		78.3%

Figure 4: Results of experiments on the hand-built Wikipedia triplet dataset.

The top three relevant pages are saved as separate png images and the text chunks along with the asked query is saved in retrieved_context.json file:

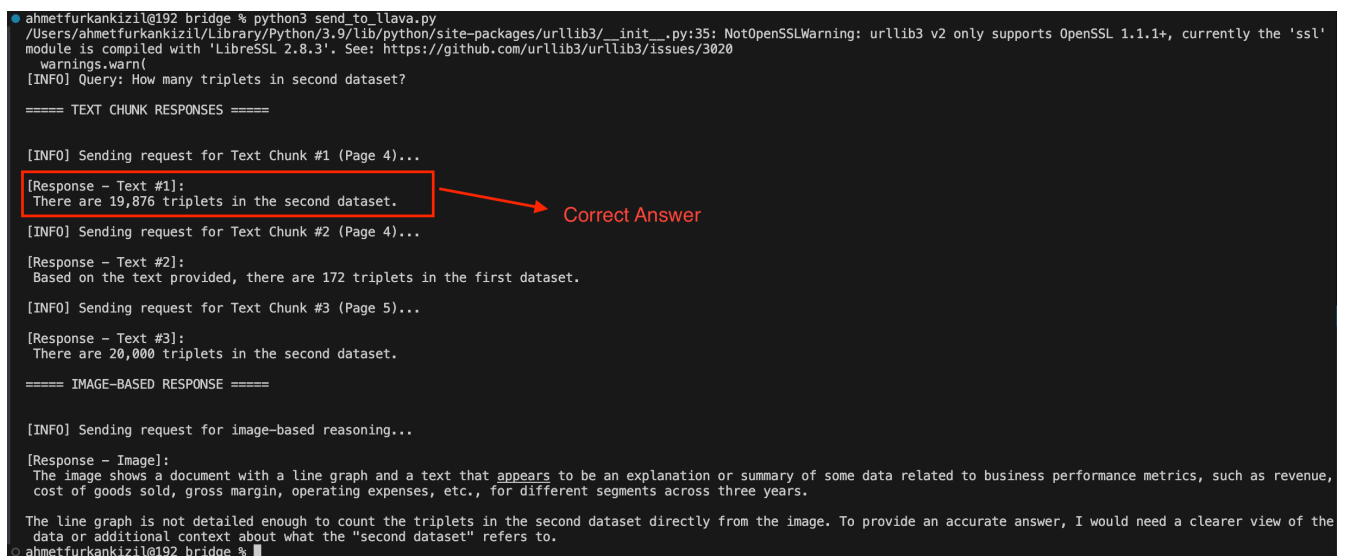


Those files will be downloaded and will be placed into the same directory where the local LLaVa model is used:



```
1 import base64
2 import json
3 import os
4 import requests
5
6 def encode_image_base64(image_path):
7     if not os.path.exists(image_path):
8         raise FileNotFoundError(f"Image {image_path} not found.")
9     with open(image_path, "rb") as f:
10         return base64.b64encode(f.read()).decode("utf-8")
11
12 def send_to_llava():
13     context_path = "retrieved_context.json"
14     if not os.path.exists(context_path):
15         raise FileNotFoundError("retrieved_context.json not found.")
16
17     with open(context_path, "r", encoding="utf-8") as f:
18         context = json.load(f)
19
20     query = context.get("query", "").strip()
21     top_texts = context.get("top_texts", [])
22     top_images = context.get("top_images", [])
23
```

Executing the `send_to_llava.py` file will use the local LLaVA model to generate answers based on the relevant text and images:



```
ahmetfurkankizil@192 bridge % python3 send_to_llava.py
/Users/ahmetfurkankizil/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl'
module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
warnings.warn(
[INFO] Query: How many triplets in second dataset?

===== TEXT CHUNK RESPONSES =====

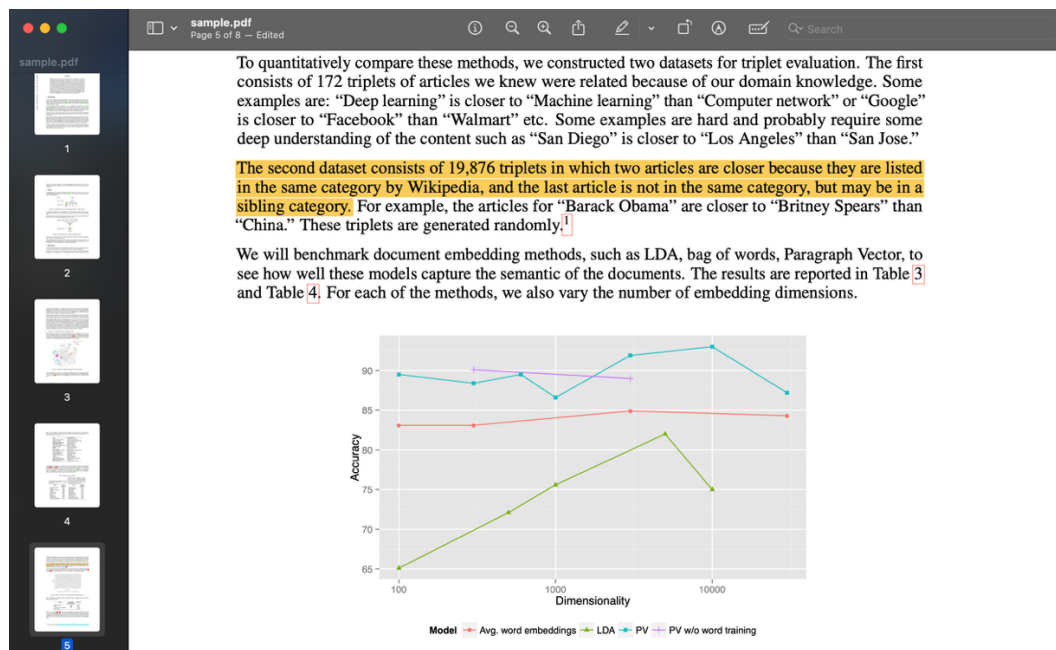
[INFO] Sending request for Text Chunk #1 (Page 4)...
[Response - Text #1]:
There are 19,876 triplets in the second dataset.
[INFO] Sending request for Text Chunk #2 (Page 4)...
[Response - Text #2]:
Based on the text provided, there are 172 triplets in the first dataset.
[INFO] Sending request for Text Chunk #3 (Page 5)...
[Response - Text #3]:
There are 20,000 triplets in the second dataset.

===== IMAGE-BASED RESPONSE =====

[INFO] Sending request for image-based reasoning...
[Response - Image]:
The image shows a document with a line graph and a text that appears to be an explanation or summary of some data related to business performance metrics, such as revenue,
cost of goods sold, gross margin, operating expenses, etc., for different segments across three years.

The line graph is not detailed enough to count the triplets in the second dataset directly from the image. To provide an accurate answer, I would need a clearer view of the
data or additional context about what the "second dataset" refers to.
ahmetfurkankizil@192 bridge %
```

The first answer is correct and the information is in the first page which is retrieved as the most relevant pages (**Page 5**):



In testing, the system successfully retrieved the correct image page containing the answer, demonstrating the strength of the retrieval mechanism. However, the response generation using the local LLaVA model fell short — often missing context or producing vague answers. This highlights current limitations in LLaVA’s reasoning depth, and suggests that integrating a stronger vision-language model (e.g., GPT-4V or Gemini Pro Vision) could significantly improve answer accuracy.

9. Conclusion

The pipeline successfully retrieved relevant text and visual content in response to real queries. While results were accurate in many cases, some queries led to hallucinated or semantically weak matches, showing that retrieval quality can vary depending on content structure and query clarity.

Next steps could include:

- Run benchmark evaluations on structured datasets (e.g., DocVQA, PubLayNet).
- Compare ColPali with other multimodal encoders for retrieval robustness.
- Add confidence filtering or post-retrieval validation to reduce noise.