Multimodal RAG Report

Author: Sviatoslav Stehnii

1. Objective

To build a **multimodal RAG (Retrieval-Augmented Generation)** system that enables users to ask questions about "The Batch" articles and receive responses grounded in **textual and visual** content, including **article body and images**.

2. Architecture Overview

Scraping Stage

The project uses a GhostScraper class to scrape content from The Batch articles

The scraping process:

a. Connects to MongoDB for storage

b. Crawls articles by tags using the Ghost API

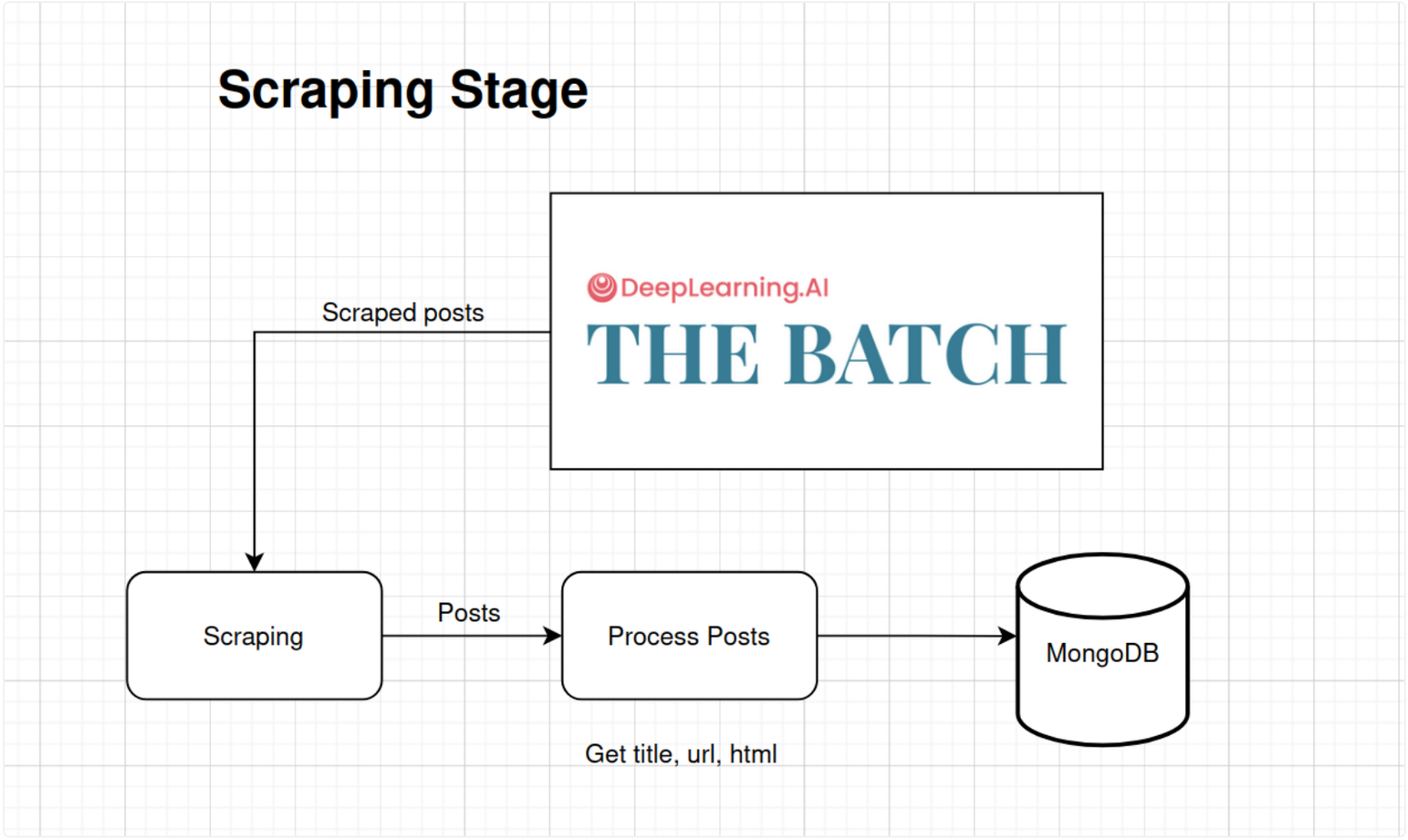
c. Processes both text content and images from articles

d. Images are downloaded, processed (resized and compressed), and stored

e. All content is stored in MongoDB with two collections:

f. articles: Stores article text content and metadata

g. images: Stores processed images and their metadata

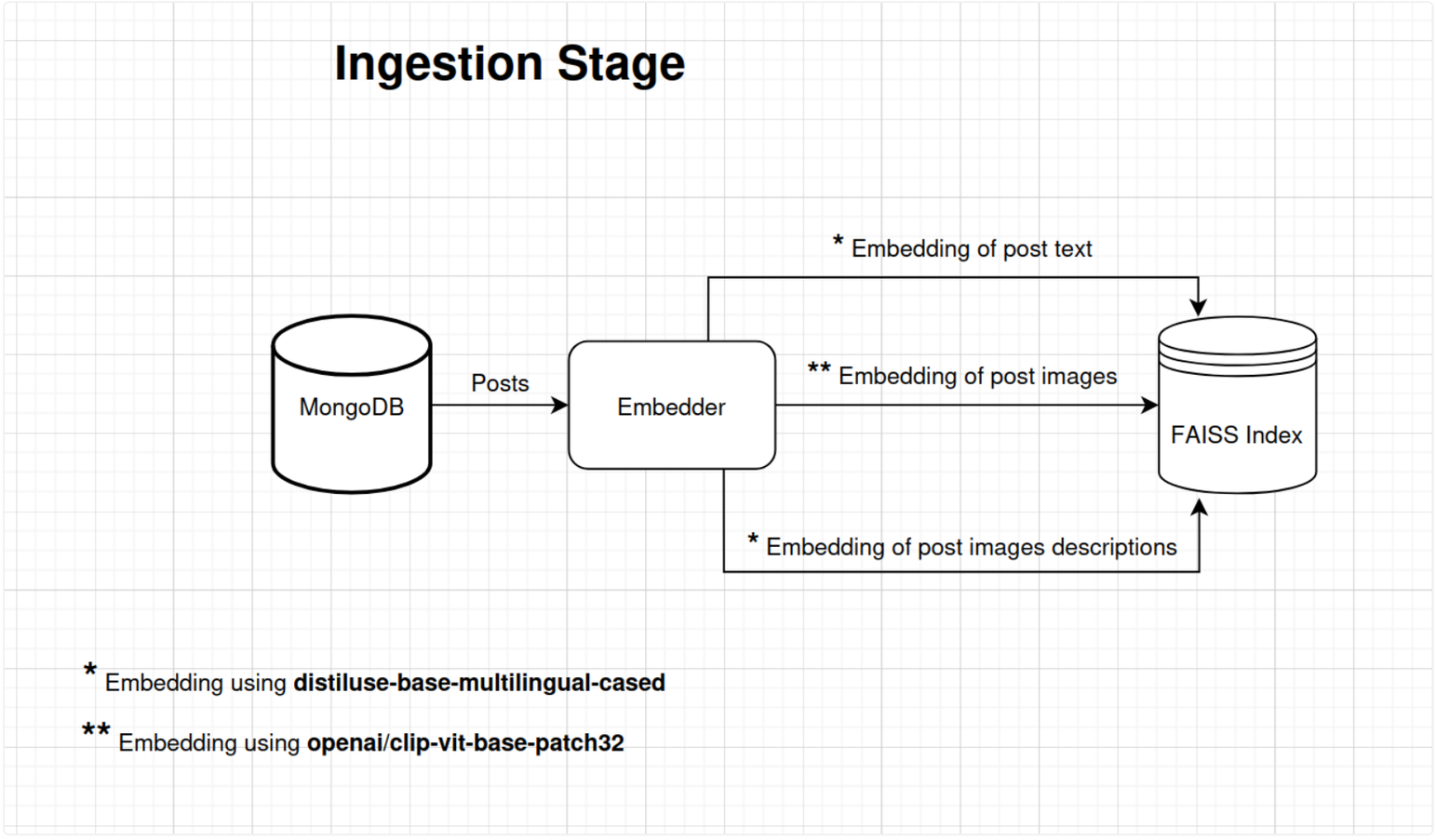


Ingestion Stage

The ingestion process is handled by the Embedder class and create_vector_store.py

Key steps:

- Text Processing:
  a. Splits articles into overlapping chunks
  b. Generates text embeddings using sentence-transformers
  c. Creates metadata for each chunk
- Image Processing:
  a. Processes images using CLIP model for embeddings
  b. Generates image descriptions using Gemini
  c. Processes alt text for images
- Vector Store Creation:
  a. Uses FAISS for efficient similarity search
  b. Creates a unified vector store for both text and image embeddings
  c. Links related chunks and images in the metadata
  d. Stores embeddings and metadata in the vector store director

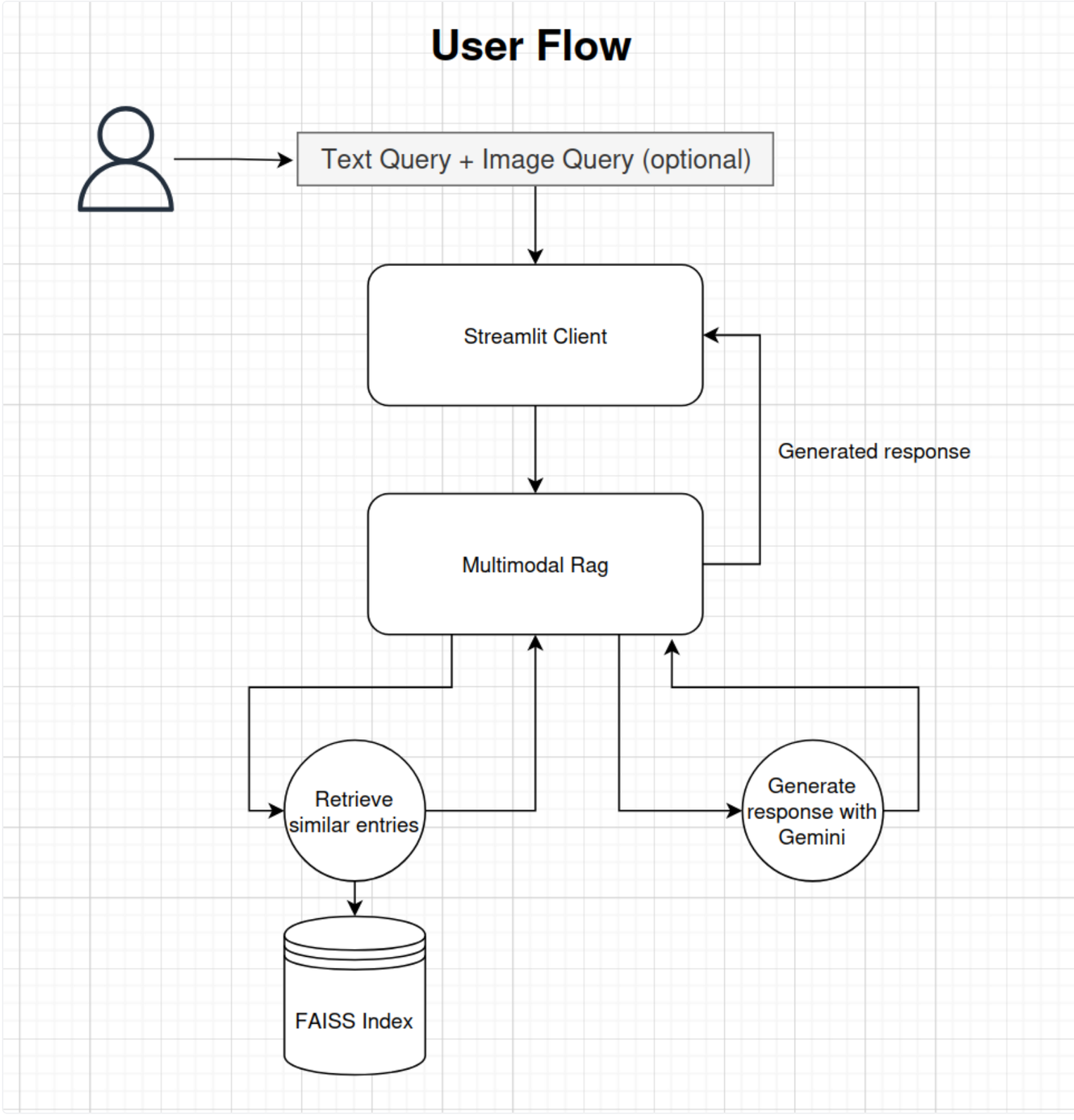


User Flow

The application provides a Streamlit web interface (app/main.py)

User interaction flow:

1. They can:
   - Enter text queries in the chat interface
   - Upload images for visual search
   - View both text and image results
2. The MultimodalRAG class handles:
   - Processing user queries (text and/or images)
   - Searching the vector store for relevant content
   - Generating responses using the retrieved content
3. Results are displayed with:
   - Generated answers
   - Source links for both text and images
   - Chat history showing the conversation flow



3. Tools & Technologies

Component	Tool/Model	Purpose
Scraping	**Ghost API**	Crawl articles, extract content
Database	**MongoDB**	Store articles, images, metadata

Image Captioning	**Gemini 2.0 Flash**	Generate image descriptions
Text Embeddings	**sentence-transformers (DistilUSE)**	Embed article content
Image Embeddings	**openai/clip-vit-base-patch32**	Embed images
Vector Search	**FAISS**	Retrieve top-k relevant items
LLM QA Model	**Gemini 2.0 Flash**	Generate final answers
UI	**Streamlit**	Frontend to interact with the system

4. Key Design Decisions & Rationale

4.1 Gemini for Image and Text

I used **Gemini (2.0 Flash)** because it's great at handling both images and text in one model. It generates image descriptions and answers questions based on retrieved content. It's fast, works well in real-time, and — for my use case — it's free or very low-cost, which made it a no-brainer for development.

4.2 MongoDB to Store Articles

MongoDB was perfect for storing everything I scraped — article text, images, and metadata — all in one place. Its flexible structure made it easy to work with different types of data, and it's fast enough for read-heavy tasks like this one.

4.3 Text & Image Embeddings

For text, I used **distiluse-base-multilingual-cased**, and for images, **CLIP (ViT-B/32)**. Both produce 512-dimensional vectors, so they're easy to compare or use side-by-side. I kept separate indexes for text and image embeddings to keep things simple and efficient.

4.4 FAISS for Search

FAISS handles the similarity search — it finds the most relevant articles or image descriptions. It's fast, works well on CPU, and fits perfectly with numpy embeddings. It's ideal for retrieving the top results for any query.

4.5 Streamlit for the UI

Streamlit made it easy to build a clean web interface. Users can ask questions, see the answers, and view related images and article snippets — all in one place. It's lightweight, fast to set up, and looks good out of the box.