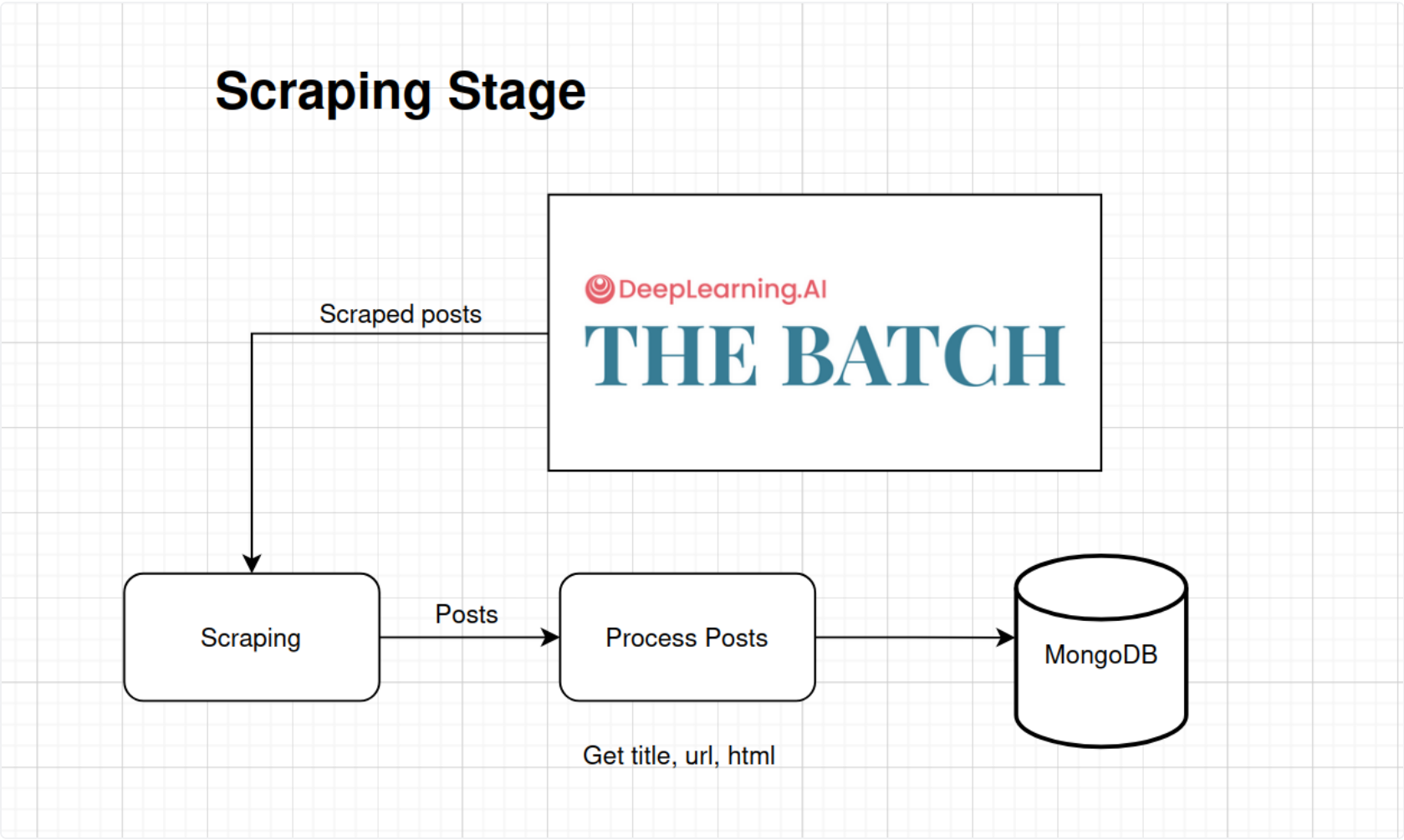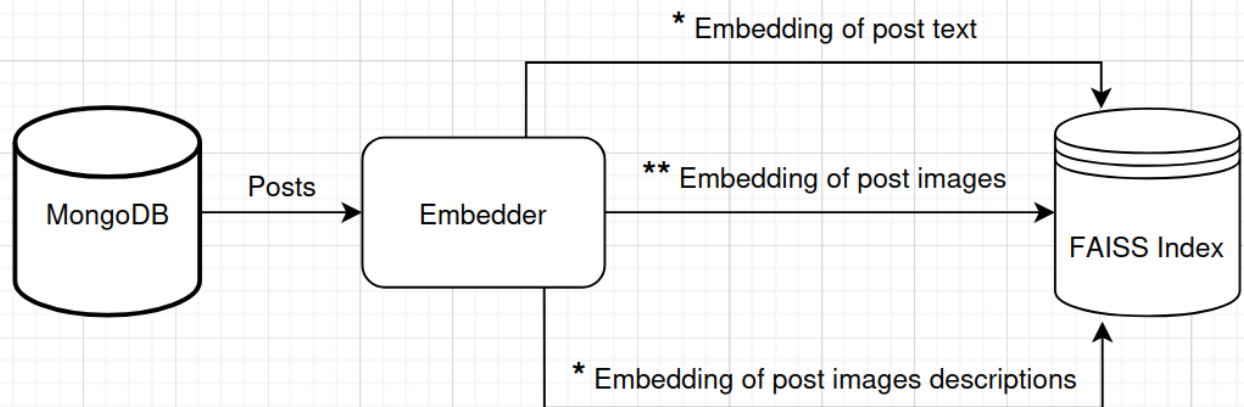# Multimodal RAG Report

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**.
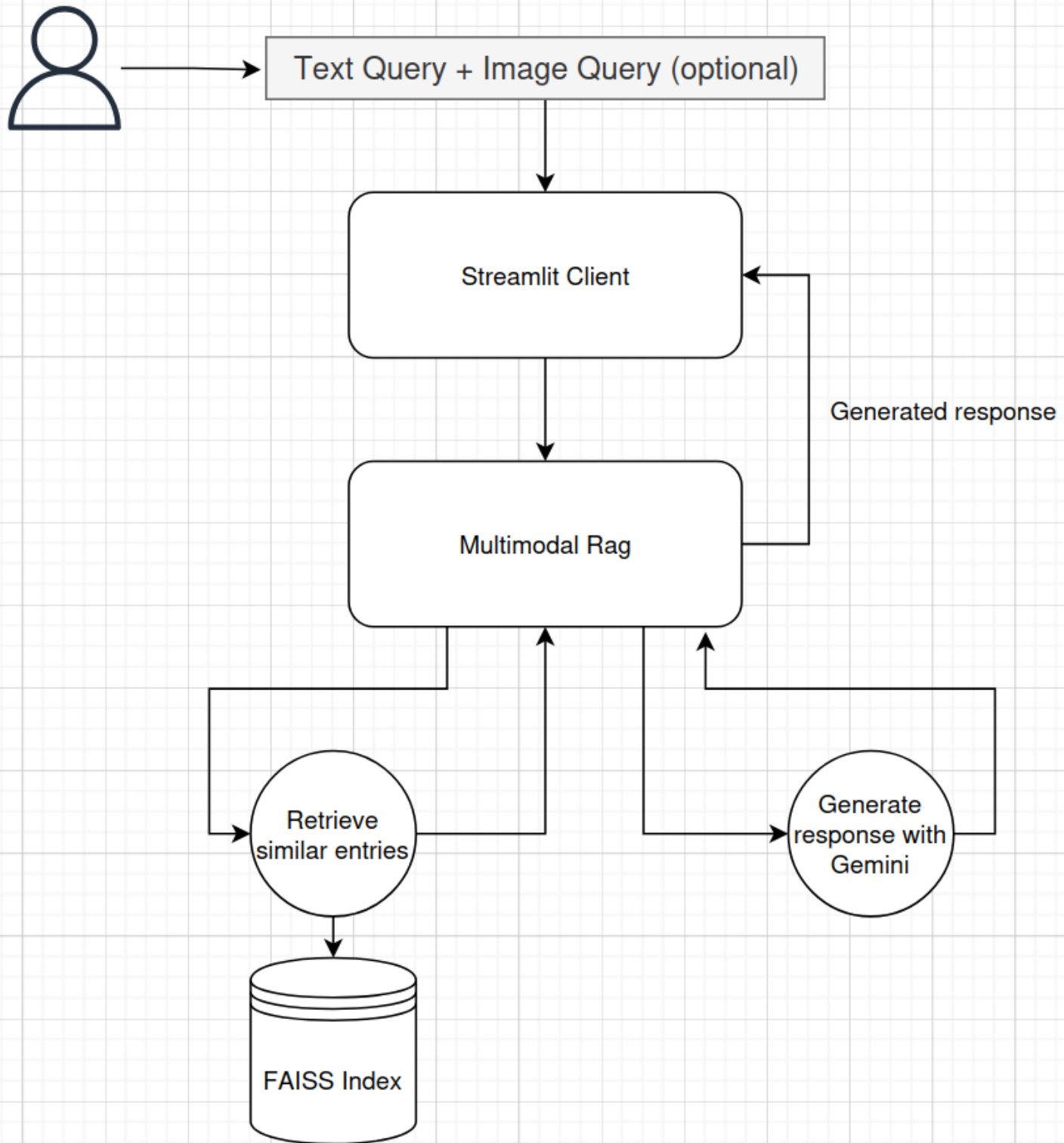
2. Architecture Overview

Ingestion Stage

MongoDB → Posts → Embedder

Embedder:
- * Embedding of post text → FAISS Index
- ** Embedding of post images → FAISS Index
- * Embedding of post images descriptions → FAISS Index

* Embedding using **distiluse-base-multilingual-cased**

** Embedding using **openai/clip-vit-base-patch32**

User 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.