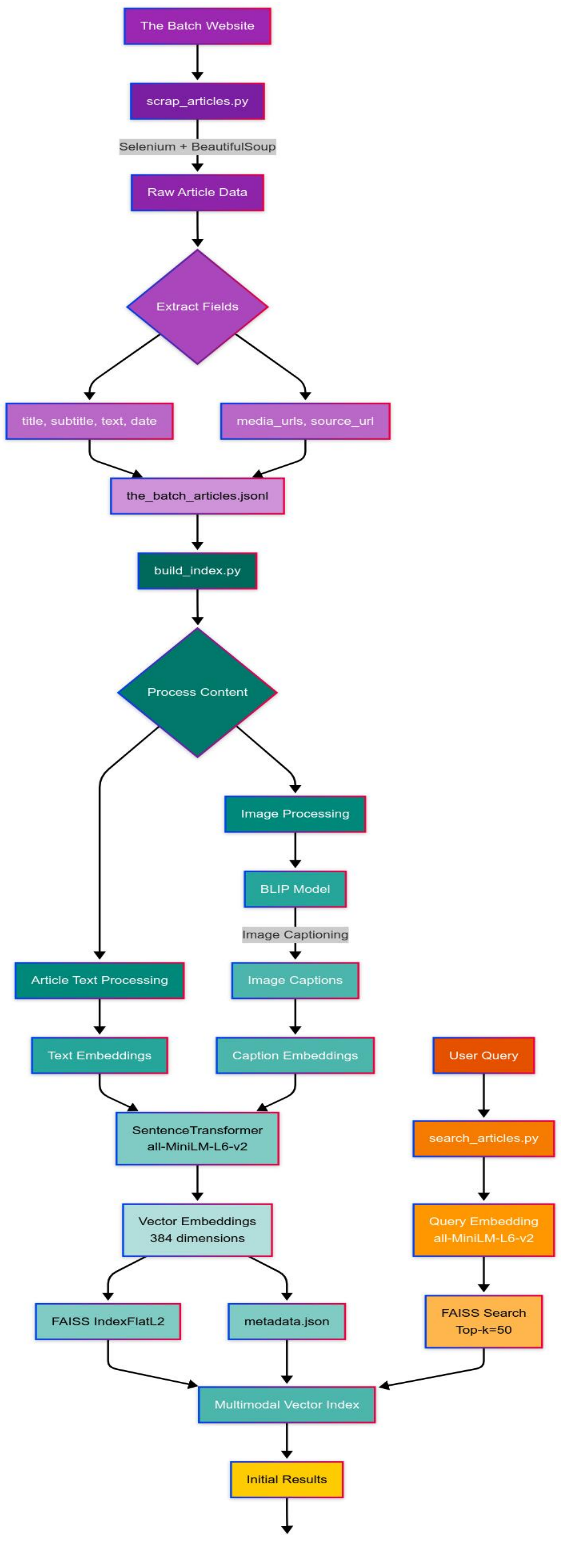
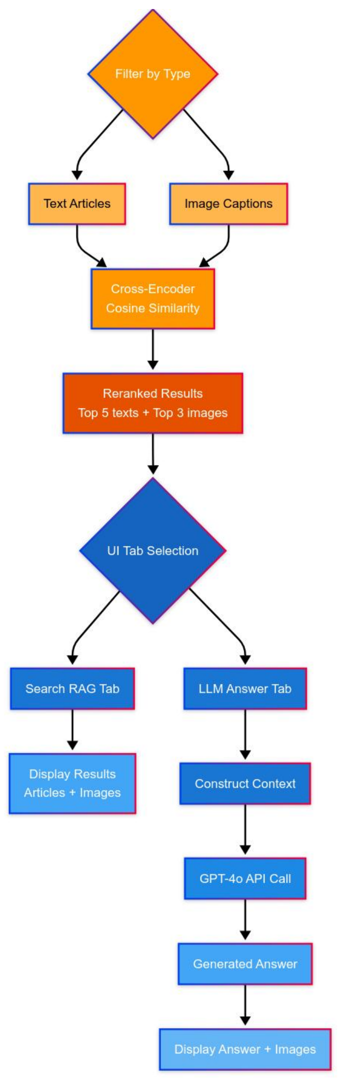
Building a Multimodal RAG System

This report outlines the design and implementation of a **Multimodal Retrieval-Augmented Generation (RAG) System** for retrieving and summarizing AI-related news articles from *The Batch*. The system integrates both **textual** and **visual** content, enabling users to query for relevant information and receive contextual answers supported by articles and images.

Data-flow for my Multimodal Rag System:





1. Data Ingestion and Preprocessing(scrap_articles.py)

The data for this project was collected from the official website of The Batch — a weekly newsletter published by DeepLearning.AI. It includes curated news articles focused on AI research, business applications, culture, and hardware.

To obtain structured article data, I developed a dedicated scraper using Selenium and BeautifulSoup. Each of the six content categories — culture, research, data points, business, science, and hardware — was crawled separately.

During scraping, the following data fields were extracted for each article:

- title: the article's main heading.
- subtitle: additional heading when available.
- text: the full content of the article, cleaned and de-duplicated.
- date: publication date.
- media_urls: direct links to embedded media (primarily images).

- source_url: the canonical link to the original article.

The final dataset was serialized as a .jsonl file (**the_batch_articles.jsonl**), where each line is a structured document containing both textual and image data for downstream indexing and retrieval.

2. Multimodal Database Creation(build_index.py)

To enable efficient retrieval across both textual and visual content, a unified multimodal vector index was constructed using the FAISS library. This index allows for fast approximate nearest neighbor search over a combination of text and image representations.
The process consisted of the following steps:

- Article and Media Structuring

Each article was represented as a single entry containing its title, subtitle, and full body text. Additionally, all related images from each article were extracted and stored as separate entries.

- Image Captioning with BLIP

Each image was processed using the BLIP image captioning model (Salesforce/blip-image-captioning-base) to generate a meaningful text description of its visual content. These captions were later embedded alongside article texts to ensure semantic alignment between modalities.

- Embedding Generation

A pre-trained SentenceTransformer model (all-MiniLM-L6-v2) was used to encode all content — article texts and image captions — into dense vector embeddings of size 384.

- Index Construction

The resulting embeddings were stacked into a NumPy matrix and passed to a FAISS IndexFlatL2 structure, which performs L2 similarity-based retrieval.

- Output

The full embedding index was saved to disk, and a metadata JSON file that preserves document types (text or image) and references to original content (e.g., article titles, URLs, and image links).

- Metadata Linking

Each entry in the index was associated with structured metadata, including:

  - article_id (to group content by source),
  - type (text or image),
  - raw_text (text content or BLIP caption),
  - meta (title, date, URL, or image URL).

This multimodal FAISS index allows for unified retrieval across textual and visual information, and serves as the backbone of the RAG (Retrieval-Augmented Generation) pipeline in later stages.

3. Search and Reranking Mechanism(search_articles.py)

To retrieve the most relevant content for a given user query, a hybrid pipeline combining approximate vector search and reranking was implemented. This mechanism ensures high recall and precision when fetching both articles and images.

Step-by-step Workflow:

Query Embedding
The user's query is first encoded into a dense vector using the same embedding model (all-MiniLM-L6-v2) that was used to build the index. This ensures consistent vector space alignment.

FAISS Search (Initial Retrieval)
A top-k (typically k=50) similarity search is performed over the FAISS index to fetch the nearest neighbors. These include both text and image entries, depending on the query.

Metadata Filtering by Type
The raw FAISS results are filtered into two groups:

Textual Articles: entries originally derived from article content.

Images: entries where the raw text is a caption generated by the BLIP model.

Reranking with Cross-Encoder Cosine Similarity
Since FAISS only performs approximate nearest neighbor search based on L2 distance, a second stage reranking is applied for higher precision:

Each result is re-encoded into embeddings.

Cosine similarity between the query and each result is computed using sentence_transformers.util.pytorch_cos_sim.

The top results are sorted by cosine score.
From the reranked results the top n text articles and top m image captions are returned (5 and 3 respectively). These results are then passed to either the user interface (for direct viewing) or to a language model (for question answering).

4. Streamlit UI Implementation with LLM Integration

To make the multimodal system accessible and easy to test, I developed an interactive user interface using Streamlit. The application provides two primary views: one focused on retrieval (text and images), and the other on answering user questions via a language model. The interface is clean, responsive, and organized into two tabs — "Search (RAG)" and "LLM Answer."

- In the first tab, users can input a query and immediately see the top-ranked results from the FAISS index. These include both article texts and image captions, which are retrieved using approximate nearest neighbor search and further reranked for semantic relevance. The interface also allows users to filter the content type — showing only text, only images, or both. Each article preview includes its title, a snippet of its content, and a link to the full source. Image results are displayed with their generated BLIP captions. If no relevant items are found, the system gracefully informs the user instead of failing silently.

- The second tab brings in natural language reasoning capabilities using a large language model. When a user ask a question, the system first performs a retrieval step similar to the first tab, but instead of displaying results directly, it constructs a structured context by combining both textual content and image captions. This context is passed, along with the user's question, to OpenAI's GPT-4o using the chat completion API. The model processes the prompt and returns a synthesized answer, which is then rendered in the UI. Any referenced images are also shown beneath the answer for visual grounding.

Through this interface, the user can explore the capabilities of the RAG pipeline and observe how retrieval and generation work together — offering both factual retrieval and contextual reasoning in one streamlined web application.

5. Evaluation with RAGAS

To evaluate the performance of the multimodal RAG system, I used the RAGAS framework, which provides a set of fine-grained metrics to assess the quality of retrieval-augmented generation. I generated synthetic questions and reference answers from randomly sampled articles using GPT-4o. These were then used to simulate a real QA pipeline and evaluate how well the system retrieves relevant context and produces faithful answers.

The evaluation was performed using the following key RAGAS metrics:

- Faithfulness – measures if the generated answer is grounded in the retrieved context.
- Answer Relevancy – checks if the answer actually addresses the question.
- Context Recall – captures how much of the ground-truth reference information is present in the retrieved context.
- Context Precision – penalizes irrelevant or unnecessary context.
- Answer Correctness – compares the generated answer to the known reference answer.
- Answer Similarity – evaluates the semantic closeness between generated and reference answers.

Metric	Score
Faithfulness	~0.92
Answer Relevancy	~0.90
Context Recall	~0.84
Context Precision	~0.79
Answer Correctness	~0.89
Answer Similarity	~0.86

All results are in **output/ragas_eval_results.csv**

The results suggest that the system performs well in generating answers that are both relevant and grounded. The faithfulness and relevancy scores are strong, indicating that the answers generated by the model closely follow the retrieved content and meaningfully address the questions. The context recall score shows that most of the important information is being retrieved, although context precision is slightly lower, pointing to room for improvement in filtering out less useful content. Overall, both answer correctness and similarity confirm that the answers are accurate and semantically aligned with the expected outputs.

Possible Improvements:

First, incorporating a **cross-encoder reranker** (e.g., ColBERT or a BERT-based model) could significantly improve answer relevance by re-evaluating candidate passages with deeper context understanding.

Second, to improve multimodal quality, **BLIP captions could be augmented** or replaced with more task-specific visual encoders (e.g., image classification or OCR models for screenshots and infographics).

On the UX side, adding **user feedback mechanisms** would help identify irrelevant results and refine retrieval over time. Finally, adopting a **better scraping strategy** (e.g., integrating with RSS feeds or using snapshot APIs) would make data collection more reliable across changes in the source website.

Usage:

App is available on https://sofibrezden-multimodal-rag-appapp-s372hk.streamlit.app/
Demo video (gif) is in ***demo_rag.gif*** in this rep.