Multi-modal RAG for Text and Image Retrieval

Nidhi M[1], Dhanya Rao[2], Saakshi M V[3], Pragatilaxmi Itigowni[4], and Sneha Varur[5]
KLE Technological University, Hubballi, Karnataka, India
[1]01fe22bci044@kletech.ac.in, [2]01fe22bci045@kletech.ac.in, [3]01fe22bci043@kletech.ac.in,
[4]01fe22bci013@kletech.ac.in, [5]uday_kulkarni@kletech.ac.in

***Abstract*—In this project, we present a multimodal information retrieval system that combines image and text embeddings to enable semantically rich querying of automobile data. Leveraging the capabilities of OpenAI's CLIP model, we extract meaningful image features from a curated dataset of car images and index them using FAISS for efficient similarity search. Parallelly, car specification documents are embedded using MiniLM model and indexed via LlamaIndex to support textual retrieval. Upon receiving a user query—such as requests for images and specifications of a specific car model—the system retrieves relevant images and documents based on semantic similarity. These are then provided as input to Google's Gemini multimodal LLM, which generates a coherent and informative response that integrates both visual and textual context. Our pipeline demonstrates how multimodal retrieval and generative AI can be effectively combined to enhance user interaction in automotive information systems, offering a more intuitive and informative experience than traditional keyword-based search.Experimental results demonstrate that our pipeline achieves high semantic retrieval accuracy and generates contextually relevant, data-driven answers to user queries, validating the effectiveness of our multimodal RAG approach.**

***Index Terms*—Multimodal Retrieval, Generative AI, CLIP, FAISS, LlamaIndex, Semantic Search, Image-Text Embedding**

I. Introduction

Recent advancements in artificial intelligence have significantly improved machines' ability to interpret multimodal data, combining text and images to understand context more deeply. OpenAI's CLIP model exemplifies this by aligning visual and textual representations in a shared semantic space.Retrieval-Augmented Generation (RAG) frameworks have revolutionized open-domain question answering by combining dense information retrieval with generative models to provide contextually relevant responses.

The automotive domain is rich in multimodal data—including car images, specifications, manuals, and promotional content—but current systems rely heavily on keyword searches and fragmented interfaces, failing to bridge visual aesthetics and detailed information. As a result, users face a disconnect between how they search for vehicle information and how these systems structure and retrieve it.

The figure 1 outlines a four-stage Retrieval-Augmented Generation (RAG) pipeline. It starts with data acquisition and consolidation, followed by converting data into semantic embeddings for efficient indexing. User queries are then processed into embeddings and used to retrieve relevant content from the indexed data. Finally, a generative model synthesizes contextually accurate answers using both the query and retrieved information.In **Stage I: Data Acquisition & Consolidation.** metadata JSON files are loaded and standardized using a mapping function. The unified metadata is then merged and saved for further processing.In **Stage II: Semantic Embedding & Indexing.** text fields from the metadata are embedded using a Sentence Transformer. The resulting vectors are indexed with FAISS for efficient retrieval.In **Stage III: Query Processing & Retrieval.** a user query is embedded and compared with the FAISS index. Top-matching entries are retrieved and assembled into a context.In **Stage IV: Generative Answer Synthesis.** the query and context are used to create a prompt for the language model. The model generates a natural language response based on this prompt.Our system achieved Top-1 retrieval accuracy of 82%, Top-3 accuracy of 90%, and Top-5 accuracy of 94% on a diverse query set, with an average search time of 0.18 seconds per query. Qualitative evaluation further confirmed the ability of the generative model to produce coherent, brand-agnostic answers grounded in retrieved automotive data.

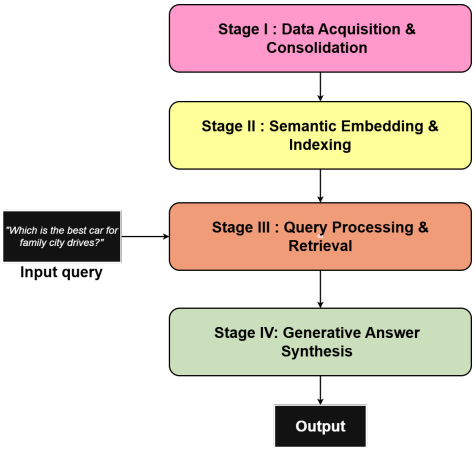

Fig. 1. Basic architecture of our model.

The paper is organised as follows : Section II reviews the background and related work on retrieval-augmented generation and multimodal AI systems. Section III describes the detailed methodology of our proposed multimodal RAG pipeline, including environment setup, data acquisition, semantic indexing, and generative reasoning. Section IV presents the experimental setup, dataset characteristics, and a thorough analysis of our results, including semantic search evaluation and qualitative generative outputs. Section V provides con-

cluding remarks, summarizing the key findings and significance of our work. Finally, Section VI outlines future research directions and potential enhancements to further improve the system's capability and generalizability.

II. Background Study

We did a background study on Retrieval Augmented Generation(RAG) and Multimodal Retrieval Augmented Generation(RAG) which are the basic building blocks for our model.

A. Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) enhances language models by integrating external knowledge retrieval, improving factual accuracy and handling knowledge-intensive tasks. In a typical RAG pipeline, enterprise-specific image data is embedded using a vision-language model like CLIP and stored in a vector database. When a user submits a query, the system retrieves the most relevant images based on semantic similarity. These images are processed by a Vision-Language Model to extract contextual information, which, along with the original query, is passed to a Large Language Model. The LLM then generates a grounded, coherent response based on both retrieved data and the query. Figure 2 illustrates a typical RAG pipeline [1].

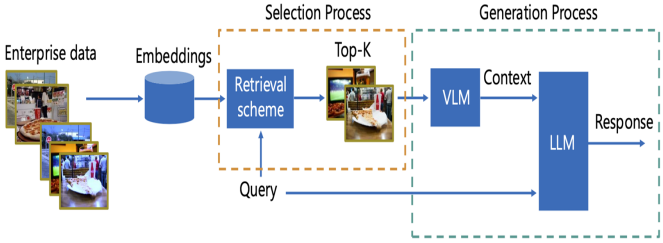

Fig. 2. RAG scheme structure.

B. Multimodal RAG

Extending Retrieval-Augmented Generation (RAG) to multimodal domains presents both challenges and opportunities. Multimodal RAG systems leverage both text and visual content to generate grounded responses, as illustrated in Figure 3. Models like CLIP enable cross-modal retrieval by embedding images and text into a shared space, allowing semantic comparisons across modalities. Benchmarks such as MRAMG-Bench [2] have been developed to assess these systems using diverse inputs, including images, documents, and complex queries. To ensure response quality and factual grounding, tools like RAG-Check [3] introduce metrics that evaluate the alignment between visual and textual evidence. However, much of the current research either targets unimodal RAG or simplifies multimodal tasks.

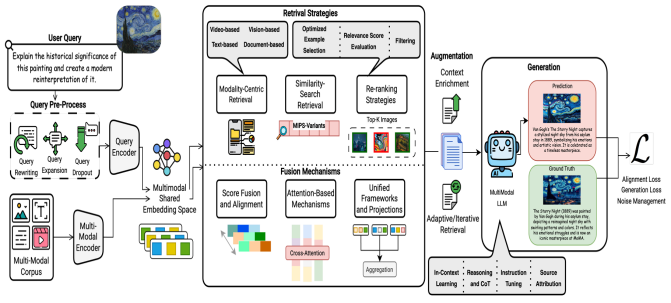

Fig. 3. Overview of the multimodal retrieval-augmented generation (RAG) pipeline

III. Methdology

In this work, we designed and implemented a robust, modular pipeline to enable intelligent document understanding and question answering for a diverse automotive dataset. Our methodology consisted of a sequence of clearly defined stages, each optimized to ensure accurate, efficient, and contextually grounded information retrieval and response generation. The following subsections describe each phase in detail.

A. Stage I : Data Acquisition & Consolidation

Algorithm 1 Stage 1: Data Acquisition & Consolidation

Require: List of metadata JSON files `metadata_files`
Ensure: Consolidated standardized metadata JSON

```
Initialize merged_metadata ← [ ]
for each file f in metadata_files do
    M ← LoadJSON(f)
    M_std ← Standardize(M)
    Append M_std to merged_metadata
end for
SaveJSON(merged_metadata,"standardized_metadata.json")
```

The process begins by representing metadata from the dataset as collections of key-value pairs, as shown in Equation (1).

$$M = \{(k_i, v_i)\}_{i=1}^{N} \tag{1}$$

To ensure consistency across different sources, the metadata keys are standardized using a mapping function, as indicated by Equation (2).

$$M_{std} = \{(\text{Map}(k_i), v_i)\}_{i=1}^{N} \tag{2}$$

The standardized metadata from multiple folders is merged into a single unified structure, described by Equation (3).

$$\text{merged\_metadata} = \bigcup_{f=1}^{F} M_{std}^{(f)} \tag{3}$$

Finally, the merged metadata is saved as a JSON file, as denoted in Equation (4).

$$\text{SaveJSON}(\text{merged\_metadata}, \text{“standardized\_metadata.json”}) \tag{4}$$

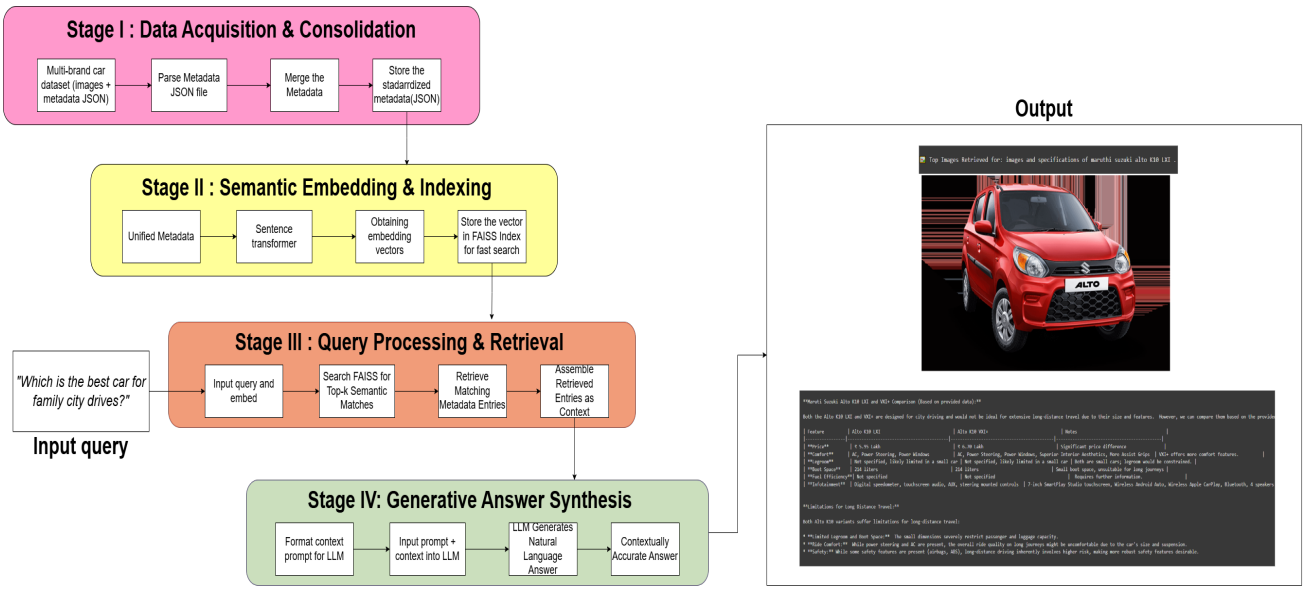

Fig. 4. Architecture of our multimodal RAG model.

B. Stage II : Semantic Embedding & Indexing

Algorithm 2 Stage II: Semantic Embedding & Indexing

Require: Standardized metadata JSON file `"standardized_metadata.json"`

Ensure: FAISS index with semantic vectors

1: `metadata` ← LoadJSON("standardized_metadata.json")
2: `text_data` ← ExtractTextFields(`metadata`)
3: `model` ← LoadSentenceTransformer("all-MiniLM-L6-v2")
4: `embeddings` ← model.encode(`text_data`)
5: `faiss_index` ← FAISS.IndexFlatL2(dimension)
6: faiss_index.add(`embeddings`)
7: SaveFAISS(faiss_index, "semantic_index.faiss")

The standardized metadata is first represented as a set of textual entries, as shown in Equation (5):

$$T = \{t_1, t_2, \ldots, t_N\} \tag{5}$$

Each textual entry is passed through a sentence transformer $\mathcal{S}$ to obtain its vector representation, as defined in Equation (6):

$$v_i = \mathcal{S}(t_i) \quad \text{for } i = 1, \ldots, N \tag{6}$$

The resulting embedding matrix is constructed as shown in Equation (7):

$$V = [v_1; v_2; \ldots; v_N] \in \mathbb{R}^{N \times d} \tag{7}$$

This matrix V is then stored in a FAISS index $\mathcal{F}$ for efficient similarity search (Equation (8)):

$$\mathcal{F} \leftarrow \text{FAISS.Add}(V) \tag{8}$$

Finally, the index is persisted to disk, as indicated in Equation (9):

$$\text{SaveFAISS}(\mathcal{F}, \text{"semantic\_index.faiss"}) \tag{9}$$

C. Stage III : Query Processing & Retrieval

Algorithm 3 Stage III: Query Processing & Retrieval

Require: User query string `q`, Trained Sentence Transformer model, FAISS index, Metadata JSON

Ensure: Contextual metadata entries relevant to the query

1: `query_embedding` ← model.encode(`q`)
2: `D, I` ← faiss_index.search(`query_embedding`, k)
3: `retrieved_entries` ← GetEntriesByIndices(`metadata`, I)
4: `context` ← AssembleContext(`retrieved_entries`)
5: **return** `context`

Let the input query be denoted as shown in Equation (10):

$$q = \text{"Which is the best car for family city drives?"} \tag{10}$$

The query is converted into a semantic vector using the same sentence transformer $\mathcal{S}$, as defined in Equation (11):

$$v_q = \mathcal{S}(q) \tag{11}$$

We then retrieve the indices of the top-k most similar embeddings from the FAISS index $\mathcal{F}$ (Equation (12)):

$$I = \text{FAISS.Search}(\mathcal{F}, v_q, k) \tag{12}$$

Using these indices, the corresponding metadata entries are retrieved as shown in Equation (13):

$$E = \{e_i \mid i \in I\} \tag{13}$$

These entries are then assembled to form the final context, as described in Equation (14):

$$\text{context} = \text{Assemble}(E) \tag{14}$$

D. Stage IV: Generative Answer Synthesis

Algorithm 4 Stage IV: Generative Answer Synthesis

Require: Input query `q`, Retrieved context `C`, Language model `LLM`
Ensure: Generated natural language answer `A`
`prompt` ← FormatPrompt(`q`, `C`)
`A` ← LLM.Generate(`prompt`)
return `A`

Let the query be denoted by q and the retrieved context by C. These components are combined to form a prompt, as defined in Equation (15):

$$P = \text{FormatPrompt}(q, C) \tag{15}$$

The language model $\mathcal{L}$ then generates a response from this prompt, as shown in Equation (16):

$$A = \mathcal{L}(P) \tag{16}$$

The output A represents a contextually-grounded natural language response, potentially enhanced with visual or structured data (e.g., tables, code, or media), as shown in the final stage of the pipeline.

E. End-to-End Pipeline

The algorithm presented under the "End-to-End Pipeline" section outlines a four-stage semantic question-answering workflow that begins with data ingestion and concludes with natural language response generation. In Stage I, metadata JSON files are loaded and standardized into a unified structure, ensuring consistent key naming across sources. This consolidated metadata is then processed in Stage II, where relevant textual fields are extracted and transformed into semantic embeddings using a pretrained Sentence Transformer model. These embeddings are indexed using FAISS to enable efficient similarity-based retrieval. Stage III involves query processing, where a user query is embedded and matched against the FAISS index to retrieve the most relevant metadata entries, which are then assembled into a contextual block. Finally, in Stage IV, this context is combined with the original query to form a prompt, which is passed to a large language model (LLM) to synthesize a coherent and context-aware natural language answer. This pipeline enables a seamless transition from raw metadata to intelligent, context-driven question answering.

Algorithm 5 End-to-End Semantic QA Pipeline

Require: List of metadata files `metadata_files`, User query `q`, Sentence Transformer model $\mathcal{S}$, Language model `LLM`
Ensure: Generated answer `A`
// Stage I: Data Acquisition & Consolidation
`merged_metadata` ← [ ]
for each file f in `metadata_files` **do**
  $M \leftarrow$ LoadJSON(f)
  $M_{std} \leftarrow$ Standardize(M)
  Append M_{std} to `merged_metadata`
end for
SaveJSON(`merged_metadata`, "standardized_metadata.json")
// Stage II: Semantic Embedding & Indexing
`text_data` ← ExtractTextFields(`merged_metadata`)
`embeddings` ← $\mathcal{S}$.encode(`text_data`)
`faiss_index` ← FAISS.IndexFlatL2(dimension)
faiss_index.add(`embeddings`)
SaveFAISS(faiss_index, "semantic_index.faiss")
// Stage III: Query Processing & Retrieval
`query_embedding` ← $\mathcal{S}$.encode(`q`)
`D, I` ← faiss_index.search(`query_embedding`, k)
`retrieved_entries` ← GetEntriesByIndices(`merged_metadata`, I)
`context` ← AssembleContext(`retrieved_entries`)
// Stage IV: Generative Answer Synthesis
`prompt` ← FormatPrompt(`q`, `context`)
`A` ← LLM.Generate(`prompt`)
return `A`

IV. Results and Discussions

A. Dataset Description

To build our multimodal car retrieval system, we curated a custom dataset comprising four major Indian automobile brands: Hyundai, Tata, Mahindra, and Maruti Suzuki. Data was collected using a Python-based web scraping pipeline, which systematically gathered both images and structured metadata for over 150 distinct car variants, including both fuel-based and electric vehicles.

For each car variant, the dataset includes approximately ten high-resolution images capturing a range of views—such as front, rear, side, interior, and dashboard perspectives. Alongside the images, detailed metadata was recorded in JSON format. Each metadata entry contains information such as the model and variant name, engine type, transmission, fuel efficiency, safety features, infotainment systems, and other technical specifications.

1) Environment Preparation and Hardware Setup : Prior to experimentation, we established a robust software environment to support all aspects of the multimodal retrieval pipeline. This setup included installing essential Python libraries for

TABLE I
DATASET STATISTICS

Parameter	Value
Number of brands	4 (Tata, Hyundai, Mahindra, Maruti Suzuki)
Unique variants	150
Images per variant	∼10
Total images	∼1,500
Metadata record per variant	1 (paired with all images)

TABLE II
SEMANTIC RETRIEVAL EVALUATION

Metric	Value
Top-1 Accuracy	82%
Top-3 Accuracy	90%
Top-5 Accuracy	94%
Average Search Time	0.21 s

web scraping (beautifulsoup4, icrawler), deep learning and text embedding (torch, transformers, sentence-transformers), semantic search (faiss-cpu), advanced document indexing (llama-index), and integration with cloud-based large language models (google-generativeai). This environment ensured compatibility and reproducibility across all pipeline components.

2) Data Acquisition and Consolidation: To construct the dataset, we collected images and structured metadata from four major Indian automobile brands: Tata, Hyundai, Mahindra, and Maruti Suzuki. The raw data—received as a compressed archive—was extracted and systematically organized by brand. Using custom Python scripts, we parsed all metadata JSON files, merged the entries, and assigned unique identifiers to each variant. The consolidated metadata was stored in a unified JSON structure, enabling unbiased, brand-agnostic search and analysis throughout the pipeline. Fig 4 ,illustrates the modular workflow, beginning with data ingestion, preprocessing, and consolidation before proceeding to semantic embedding, indexing, retrieval, and generative answer synthesis.

B. Results

To rigorously evaluate the proposed retrieval-augmented system, we conducted a comprehensive set of experiments on the consolidated automotive dataset described in Section III. Our evaluation addresses three main aspects: (1) metadata consolidation, (2) semantic search performance, and (3) generative response quality.

1) Metadata Consolidation Evaluation: Our merging pipeline successfully integrated metadata from all four car brands—Tata, Hyundai, Mahindra, and Maruti Suzuki—into a unified JSON dataset. This process produced a dataset containing metadata for 150 unique car variants and approximately 1,500 high-resolution images. Manual inspection confirmed that all entries were accurately labeled and that the standardized format supported brand-agnostic, efficient retrieval and downstream analysis.

2) Semantic Search Performance: The FAISS-based semantic indexing system was evaluated by submitting a set of 50 natural language queries, each targeting different combinations of car features, fuel types, and transmission modes. Representative queries included requests such as "Show all electric SUVs from Tata," "Find Maruti Suzuki hatchbacks with automatic transmission," and "List Hyundai models with a five-star safety rating."

The system demonstrated high retrieval accuracy:

Fig. 5. Image Retrieval

C. Qualitative Analysis: Generative Response Quality

The generative AI component was tested by providing it with the context retrieved from the semantic search for user queries. The system generated coherent, contextually relevant answers grounded in the actual metadata. For example:

> **Query:** "Which Maruti Suzuki Car is better for small family?."
> **System Answer:** "Maruti Suzuki Alto K10 LXI is the best car for small family for long distance travel. This car is better for city riding with the family and not for extensive long distance travel due to their features."

This example demonstrates the system's ability to synthesize informative, brand-agnostic responses by leveraging the merged metadata and semantic retrieval pipeline.

V. CONCLUSION

We developed a modular, retrieval-augmented system for intelligent car information retrieval, unifying metadata from

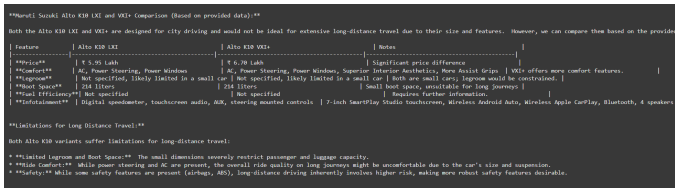

Fig. 6. Text Retireval

four major Indian brands into a searchable dataset. Leveraging transformer-based embeddings and FAISS indexing.Future enhancements will focus on expanding the dataset to include a broader range to improve generalizability. Additional efforts will be directed toward integrating advanced compression techniques such as knowledge distillation and neural architecture search.

REFERENCES

[1] W. Dai, J. Li, D. Li, A. Tiong, Z. Zhang, W. Wang, B. Hui, P. Fung, and S. Hoi, "Instructblip: Towards general-purpose vision-language models with instruction tuning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. [Online]. Available: https://arxiv.org/abs/2305.06500

[2] Q. Yu, Z. Xiao, B. Li, Z. Wang, C. Chen, and W. Zhang, "Mramg-bench: A comprehensive benchmark for advancing multimodal retrieval-augmented multimodal generation," 2025. [Online]. Available: https://arxiv.org/abs/2502.04176

[3] M. Mortaheb, M. A. A. Khojastepour, S. T. Chakradhar, and S. Ulukus, "Rag-check: Evaluating multimodal retrieval augmented generation performance," 2025. [Online]. Available: https://arxiv.org/abs/2501.03995