A Self-Correcting Multimodal RAG Framework for Iranian Tourism with Active Retrieval and Selective Fact-Checking

Amir Malekhosseini
Sharif University of Technology
amirmalekhosseini1@gmail.com

Saeed Zeynalpour
Sharif University of Technology
Fastrongs@gmail.com

Parniya Seifi
Sharif University of Technology
parniya.seifi1995@gmail.com

Tala Kiyani
Sharif University of Technology
tala.kiani78@gmail.com

Amin Fadaee
Sharif University of Technology
Fadaie.amin98@gmail.com

Abstract

Large Vision-Language Models (LVLMs) demonstrate impressive generative capabilities but often lack domain-specific knowledge and are susceptible to performance degradation from irrelevant contextual information. To address these limitations, we present a novel multimodal Retrieval-Augmented Generation (RAG) framework tailored for the specialized domain of Iranian tourism. Our system integrates a hybrid retrieval architecture, capable of processing both textual and image-based queries, by leveraging fine-tuned Sentence-Transformer and CLIP models. The core of our contribution is a generator built upon the `Qwen/Qwen2.5-VL-7B-Instruct` model, which we enhance through two primary innovations. First, we employ a **SURf (Selective Use of Retrieved Facts)** [4] fine-tuning strategy on a custom-built dataset to train the model to discern and disregard distractor information. Second, we introduce a dynamic generation process inspired by **FLARE (Forward-Looking Active Retrieval)** [2], where the model actively seeks additional information when its confidence is low. Uniquely, our system utilizes the LLM's own reasoning for meta-cognitive tasks, including query intent classification, response confidence validation, and dynamic in-context example generation. This work demonstrates a practical and effective method for creating more autonomous, robust, and context-aware RAG systems.

1 Introduction

The advent of Large Language Models (LLMs) and, more recently, Large Vision-Language Models (LVLMs), has revolutionized natural language understanding and generation. However, their knowledge is confined to their training data, rendering them less effective for tasks requiring real-time, specialized, or niche information. Retrieval-Augmented Generation (RAG) [1] has emerged as a leading paradigm to overcome this limitation by augmenting the model's internal knowledge with external, relevant documents retrieved from a corpus.

While RAG systems enhance factual accuracy, they introduce new challenges. As demonstrated by Yin et al. (2023), the performance of LVLMs can be significantly compromised by irrelevant or contradictory information in the retrieved context [3]. Furthermore, the conventional RAG pipeline is often static: information is retrieved once based on the initial query and then fed to the generator. This "retrieve-then-read" approach fails if the initial retrieval is insufficient, as the model has no mechanism to request clarification or additional details.

To address these challenges, we have developed a comprehensive multimodal RAG system specifically for the domain of Iranian tourism. Our work integrates and extends state-of-the-art techniques to create a more intelligent and resilient question-answering agent. The primary contributions of this project are:

- **A Hybrid, Multimodal Retrieval Engine:** Our system can process both textual and visual queries to retrieve relevant information from a custom-built knowledge base, making it highly versatile for real-world tourism applications.
- **SURf-Enhanced Robustness:** We implement the training methodology proposed by the SURf paper [4] to fine-tune our generator. This teaches the model to selectively use retrieved information and ignore "distractor" documents, thereby improving its resilience to imperfect retrieval results.
- **An LLM-Powered Active Retrieval Loop:** Inspired by FLARE [2], our system uses an iterative process for complex queries. Crucially, we leverage the LLM itself to perform meta-cognitive functions: it assesses its own confidence in a generated sentence and, if uncertain, formulates a new query to retrieve supplementary context before continuing. This "self-judging" capability marks a significant deviation from standard confidence-scoring methods.
- **Dynamic In-Context Example Generation:** To further improve generation quality, our system dynamically constructs a one-shot example from the retrieved content and injects it into the final prompt, effectively teaching the model the desired output format on-the-fly.

2 Methodology

Our system architecture consists of three main stages: Data Preparation, Retriever and Generator Fine-Tuning, and an Intelligent RAG Framework that directs the query process.

2.1 Dataset and Corpus

The project's knowledge base was curated to cover tourist, historical, and cultural attractions across Iran. The dataset consists of **1061 unique locations** from **17 provinces**, supported by a total of **4451 associated images**. Data for each location was structured in JSON format, containing its name, a detailed description, and a list of corresponding image paths. A crucial preprocessing step involved programmatically verifying and correcting all image paths to ensure their validity. Furthermore, to enhance the text retriever's performance, the textual descriptions were augmented to explicitly include the name of each location, thereby creating a stronger semantic link for embedding generation.

2.2 Retriever Component

The retriever module is designed to handle both text and image queries using a hybrid approach. The embeddings for both modalities are indexed using FAISS (Facebook AI Similarity Search) for efficient retrieval.

2.2.1 Text Retriever

For semantic text matching, we employed a sentence-transformer model. To adapt the model to the specific linguistic nuances of the tourism domain, it was first fine-tuned using a **contrastive learning** objective. This training was performed on a curated subset of our data, consisting of **724 locations from 10 provinces**. After this domain-specific training, the enhanced model was used to generate embeddings for the "name" and "description" fields of the **entire corpus of 1,061 locations**. These embeddings were then indexed in a FAISS `IndexFlatIP` database for efficient similarity search.

2.2.2 Image Retriever

For content-based image retrieval, a CLIP model (`openai/clip-vit-base-patch32`) was used. To improve its understanding of the specific visual features of Iranian architecture and landscapes, the model was also fine-tuned using a **contrastive learning** approach. This training was conducted on the same curated subset of **724 locations from 10 provinces** and their associated images. Following this specialization, the fine-tuned CLIP model was used to generate embeddings for all **4,451 images** in the complete dataset, which were subsequently indexed in a FAISS `IndexFlatL2` database.

2.3 Generator Component: Fine-Tuning with SURf

To mitigate the issue of model distraction by irrelevant retrieved content [3], we adopted the Selective Use of Retrieved Facts (SURf) fine-tuning strategy [4]. This approach trains the model to differentiate between relevant and irrelevant information provided in its context window.

Model: The core of our generator is the `Qwen/Qwen2.5-VL-7B-Instruct` model, a powerful Large Vision-Language Model. It was loaded using 4-bit quantization via the `bitsandbytes` library to ensure it could be trained and run on available hardware.

Dataset Generation: We programmatically generated a synthetic training dataset tailored for this task. For each tourist location, which serves as the "positive document," a corresponding question was formulated (e.g., "Tell me about [location name]"). We then leveraged our fine-tuned text retriever to find a set of similar documents. From this set, the most dissimilar document—after being verified with fuzzy string matching to ensure it was not a variant of the same location—was selected as the "negative document" or "distractor." This process yielded a dataset where each entry consists of an image, a question, a relevant "positive document," an irrelevant "negative document," and a ground-truth answer derived from the positive document.

Fine-Tuning Process: The Qwen model was then fine-tuned on this synthetically generated dataset. The training was guided by a specialized prompt template (see Appendix 1) that explicitly instructs the model to use the information from the positive document while completely ignoring the negative one. For efficient training, we utilized the PEFT library with LoRA (Low-Rank Adaptation), specifically targeting the model's query, key, value, and output projection layers.

2.4 Intelligent RAG Framework

Our framework is designed as a dynamic, multi-pipeline system that adapts its information retrieval and generation strategy based on the user's query. This approach enhances both efficiency and the quality of the generated response. The process begins with an intent classification module that routes the query to the most appropriate pipeline.

2.4.1 Intent Classification Module

The framework's entry point is an LLM-powered intent classification module. For text-only queries, this module analyzes the user's input to determine its nature, classifying it as either:

- **`DEEP_DIVE`:** Queries seeking specific, detailed information about a single entity (e.g., "What are the features of Shah Qasim Dam?").
- **`LIST_REQUEST`:** Broad queries asking for multiple options or recommendations (e.g., "What are some places to visit in Gilan?").

This classification acts as a crucial routing mechanism, directing the request to the specialized pipeline best suited to handle it. Queries that include an image are automatically classified as `DEEP_DIVE`, as they inherently focus on a specific visual subject.

2.4.2 List Generation Pipeline

For queries classified as `LIST_REQUEST`, a straightforward RAG pipeline is executed. It retrieves the top-k most relevant documents from the text-based vector index. The content from these documents is then consolidated and presented to the LLM. Using a specific prompt (see Appendix 3), the model synthesizes this information into a concise, user-friendly, and formatted list, directly addressing the user's request for multiple items.

2.4.3 Active Retrieval Pipeline for Deep Dives

For `DEEP_DIVE` queries, the system invokes a more advanced, iterative pipeline inspired by the Forward-Looking Active REtrieval (FLARE) methodology. This multi-step process allows the model to build an answer progressively and seek out more information if needed. Each step in the generation loop acts as a refinement stage, compelling the model to build upon its previous output and incrementally converge towards the most accurate and comprehensive answer it can formulate.

1. **Initial Retrieval:** The system first performs a hybrid retrieval. If an image is provided, it uses the CLIP model to find the most visually similar entry in the image database. If the query is text-only, it uses the sentence-transformer model to find the most semantically relevant document. This initial document forms the starting context for the generation process.

2. **Dynamic Example Generation:** Before generating the main answer, the system constructs a dynamic, "one-shot" example on-the-fly. It uses the LLM to generate a simple question (Appendix 4) and a concise answer (Appendix 5) based on the retrieved document. This example is prepended to the main prompt to guide the LLM on the desired tone, length, and format of its output, a technique known as in-context learning. This process is repeated at the beginning of each generation step to provide fresh, relevant guidance.

3. **Iterative Generation and Self-Correction:** The system enters a generation loop to construct the final answer sentence by sentence, with a maximum of three steps by default. Each step consists of the following phases:

   (a) **Generate:** It produces a candidate sentence based on the user's original query and all context gathered so far (see Appendix 6).

   (b) **Self-Correction (Confidence Check):** In a key deviation from the original FLARE paper, the system performs a self-correction step. It uses a meta-prompt (Appendix 7) to explicitly ask the LLM to classify its own generated statement as either `CONFIDENT` or `UNCERTAIN`. This step is designed to detect potential hallucinations or statements not supported by the current context.

   (c) **Active Retrieval (if needed):** If the model reports `UNCERTAIN`, it signifies a knowledge gap. A subsequent **query formulation step** is triggered (Appendix 8), where the model generates a new search query based on the uncertain sentence. A new document is then retrieved using this query and appended to the existing context, enriching the information available for the next generation attempt.

   (d) **Append and Iterate:** If the model reports `CONFIDENT`, the sentence is appended to the final answer. The loop then continues to the next step, repeating the process with the updated answer and context until the maximum number of steps is reached or the model generates a short, concluding sentence.

3 Results and Analysis

To evaluate the effectiveness of our proposed FLARE-SURF framework, we conducted a comprehensive assessment using a custom-built, domain-specific dataset. The evaluation was designed to measure the performance of both the retriever and the end-to-end generator components on both familiar and novel data.

3.1 Evaluation Methodology

The evaluation was performed using a set of 100 Multiple-Choice Questions (MCQs) designed to test the system's knowledge about the locations in our corpus. Each question has a single correct answer that can be inferred from the information present in the corresponding document.

To assess the model's ability to generalize, this dataset was split into two distinct subsets:

- **Seen Data (80 MCQs):** This set consists of questions related to the 724 locations that were used during the fine-tuning process of the retriever and generator models.
- **Unseen Data (20 MCQs):** This set contains questions about the remaining 337 locations that the models did not encounter during fine-tuning, thereby testing their zero-shot generalization capabilities.

We evaluated the system on three key metrics: Retriever Answer Recall Rate, Retriever Hit Rate@2, and Generator End-to-End Accuracy.

3.2 Performance Evaluation

The results of our experiments are summarized in Table 1. Overall, the system demonstrates high performance and strong generalization capabilities.

Table 1: Performance Metrics on Seen and Unseen MCQ Datasets.

Metric	**Seen Data (80 MCQs)**	**Unseen Data (20 MCQs)**
Retriever Answer Recall Rate	85.00%	80.00%
Retriever Hit Rate@2	82.50%	80.00%
Generator End-to-End Accuracy	**81.25%**	**80.00%**

3.2.1 Retriever Performance Analysis

The retriever's primary role is to provide the generator with the correct context. We measured its effectiveness using two metrics:

- **Answer Recall Rate:** This metric measures whether the single, top-retrieved document contained the necessary information to answer the MCQ correctly. With a recall rate of **85.00%** on seen data and **80.00%** on unseen data, the retriever demonstrates a strong ability to identify the correct source document. The slight decrease on unseen data is expected but indicates a high degree of generalization.
- **Hit Rate@2:** This measures whether the correct document was present within the top 2 retrieved results. The high scores of **82.50%** and **80.00%** further confirm the retriever's reliability.

These results indicate that our fine-tuned hybrid retriever is highly effective, forming a solid foundation for the generator. In the vast majority of cases, the generator is supplied with the correct information from the very first step.

3.2.2 Generator Performance Analysis

The ultimate measure of the system's success is its ability to use the retrieved context to answer questions correctly.

- **End-to-End Accuracy:** This metric evaluates the entire framework, marking an answer as correct only if the final generated response correctly answers the MCQ. Our system achieves an impressive accuracy of **81.25%** on seen data.

Most notably, the performance on unseen data remains remarkably high at **80.00%**. This minimal drop in accuracy highlights the robustness of our approach. It suggests that the combination of SURf fine-tuning—which teaches the model to handle potentially imperfect context—and the FLARE-inspired iterative process allows the model to generalize its reasoning capabilities effectively to new, unseen entities. The model is not merely memorizing facts from its training data but is successfully applying its learned skills to novel information.

4 Conclusion and Future Work

In this project, we have successfully designed and implemented a multimodal, dynamic RAG system that leverages the reasoning capabilities of an LLM to direct its own information retrieval and generation process. By integrating a hybrid retrieval system with a generator fine-tuned using the SURf methodology, our model demonstrates enhanced robustness against irrelevant context. Furthermore, by adapting the active retrieval principles of FLARE, the system can iteratively refine its knowledge base to produce more comprehensive and accurate answers to complex, domain-specific queries.

Our approach of using the LLM for self-assessment (intent classification and confidence checking) and for generating dynamic in-context examples represents a novel step towards more autonomous and intelligent QA systems.

For future work, several promising avenues exist for further enhancement:

- **Evaluating More Powerful Generator Models:** Upgrading the core generator to a more advanced VLM could significantly enhance its ability to interpret complex prompts and would likely improve the reliability of the self-confidence assessment mechanism.
- **Enhancing Dynamic In-Context Examples:** As our work highlighted the critical importance of in-context examples, future research could focus on developing more sophisticated methods for generating these exemplars. Creating examples that are more closely aligned with the user's specific question could significantly improve the generator's focus and final answer quality.
- **Optimizing Active Retrieval Query Formulation:** The effectiveness of the active retrieval loop is highly dependent on the quality of the follow-up queries. A key area for improvement is to refine the query generation process when the model is unconfident, ensuring that new queries are more targeted at retrieving the precise information needed to resolve ambiguity.
- **Advanced Retrieval Strategies:** Beyond the active retrieval loop, exploring other techniques such as query rewriting or expansion for the initial retrieval step could improve document relevance and potentially reduce the need for multiple generation cycles.
- **Corpus Expansion and Diversification:** Incorporating a larger and more diverse set of documents about Iranian tourism would further improve the system's overall recall and knowledge coverage.
- **User Feedback Loop:** Implementing a mechanism for users to provide feedback on answers could enable continuous, reinforcement learning-based improvement of the model over time.

References

[1] Lewis, P., et al. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* arXiv preprint arXiv:2005.11401.

[2] Jiang, Z., et al. (2023). *Active Retrieval Augmented Generation.* arXiv preprint arXiv:2305.06983.

[3] Yin, Z., et al. (2023). *How Does the Textual Information Affect the Performance of Vision-Language Models?.* arXiv preprint arXiv:2306.01899.

[4] Long, O., et al. (2023). *SURF: A General Method for Editing Large Language Models? No, A Method for Selectively Using Retrieved Facts.* arXiv preprint arXiv:2312.04018.

A Appendix / System Prompts

This appendix contains the English translations of the key prompts used to control the behavior of the generator model within the intelligent RAG framework. The placeholders in the format of `{variable_name}` are dynamically populated by the Python code during runtime.

A.1 SURf Dataset Generation and Fine-Tuning Prompts

```
<|im_start|>system
You are an intelligent assistant for a tourism chatbot. Your task is to answer
    questions about the place shown in the image. You are given two pieces of text
    as context. One of them is completely relevant and correct, and the other is
    misleading or irrelevant. You must selectively use only the correct context to
    generate your response. Completely ignore the irrelevant context.<|im_end|>
<|im_start|>user
[Image attached]

Here is the context:
[Context 1 - Relevant]: {example['positive_doc']}
[Context 2 - Irrelevant]: {example['negative_doc']}

Question: {example['question']}<|im_end|>
<|im_start|>assistant
{example['ground_truth_answer']}<|im_end|>
```

Listing 1: Prompt template used to fine-tune the Qwen-VL model for the SURf (Selective Uninformative Retrievable facts) methodology.

A.2 Intent Classification and RAG Prompts

```
You are an expert query classifier. Classify the user's query into one of two
    categories:
1. DEEP_DIVE: The user is asking about a specific place, person, or thing.
   (Examples: 'Tell me about Shorabil Lake', 'Where is Shah Qasim Dam?', 'What are
       the features of Abadar Village?')
2. LIST_REQUEST: The user is asking for a list of multiple places or things.
   (Examples: 'What are some places to visit in Ardabil?', 'recommend tourist spots
       in Yasuj', 'list of historical sites')

User query: "{query}"

Classification (respond with only DEEP_DIVE or LIST_REQUEST):
```

Listing 2: Prompt to classify a user's textual query to determine the appropriate RAG pipeline (Deep Dive vs. List Request).

```
You are a helpful and engaging tour guide assistant. Based on the following
    information for several places, present them to the user in a friendly,
    numbered list format in Persian. For each place, include its name and a concise
     summary of its description. Start with a welcoming sentence relevant to the
    user's query.

User's original query: '{query}'
Information about places:
---
{formatted_details_for_llm}
---

```

```
Formatted response:
```

Listing 3: Prompt used to generate a summarized list of places for LIST REQUEST queries.

A.3 FLARE (Forward-Looking Active Retrieval) Prompts

```
You are a helpful assistant. Your task is to create a single, simple, and direct
    question in Persian that can be answered by the following text. Do not add any
    extra words, phrases, or conversational filler like 'question:'. Only return
    the question itself.

Text: "{latest_doc_for_example.get('description', '')}"

Question:
```

Listing 4: Prompt for generating a dynamic example question from retrieved text, used in the few-shot context for the main generation prompt.

```
You are a direct and factual assistant. Based ONLY on the text provided, answer the
    following question in a single, concise Persian sentence.

Text: "{latest_doc_for_example.get('description', '')}"

Question: "{example_question}"

Answer:
```

Listing 5: Prompt for generating a dynamic example answer. The model is instructed to answer concisely based only on the provided text.

```
You are a helpful assistant. Your task is to answer the user's question directly and
     concisely based ONLY on the provided context. Do not add any extra information
     that does not directly answer the question.

--- Example ---
Context: {latest_doc_for_example.get('description', '')}
User Question: {example_question}
Final Answer: {example_answer}
--- End Example ---

--- Context ---
{{context}}

--- User Question ---
{query}

--- Final Answer (Answer the user's question using ONLY the context) ---
```

Listing 6: The main generation prompt used in each iteration of the FLARE loop. It includes a dynamically generated few-shot example.

```
A user asked this question: "{original_query}"

In response, the system generated this statement: "{sentence}"

Is the generated statement a specific, confident fact that directly helps answer the
     user's question, or is it a vague, uncertain guess, unrelated to user's
    question or a wrong answer? Answer with only the word **CONFIDENT** or **
    UNCERTAIN**.
```

Listing 7: Meta-prompt for the confidence check step in the FLARE loop.

```
Extract the most important keywords or entities from the following sentence to use
    as a search query.

Sentence: "{sentence}"

Search Keywords:
```

Listing 8: Prompt for generating a search query from a low-confidence sentence in the FLARE loop.