A Comparative Analysis of Retrieval-Augmented Generation Systems for Travel Planning and Cultural Navigation in Nepal

Savyata Regmi
University of New Haven

Sregm3@unh.newhaven.edu

May 4, 2025

Abstract

This paper investigates how Retrieval-Augmented Generation (RAG) systems can enhance personalized travel planning and cultural insights for tourism in Nepal. We introduce a tailored RAG framework to integrate the retrieval of travel and cultural information with generated responses from three open-source Large Language Models (LLMs): FLAN-T5 Large, FLAN-T5 Base, and TinyLLaMA. A custom dataset was created featuring typical travel-related questions, ranging from local cuisines and trekking routes to cultural norms and navigation tips, sourced from travel blogs, user reviews, official tourism sites, and lodging platforms.

Each model was evaluated on thirteen domain-specific questions to measure relevance, factual accuracy, reasoning, and hallucination. Among them, **TinyLLaMA** delivered the most fluent and detailed answers, with the highest average relevance score **(3.77/5),** although it showed a tendency to hallucinate. Meanwhile, the **FLAN-T5** models, especially **FLAN-T5 Large (2.23/5),** were more factually grounded but often produced shorter or vague responses. These findings highlight the trade-offs between output richness, factual dependability, and model efficiency. Hence, optimizing RAG-based travel applications requires thoughtful model selection and precise prompt tuning.

1 Introduction

1.1 Motivation

As the world becomes more connected, travelers are increasingly looking for experiences that go beyond convenience, they want something personal, authentic, and rooted in local culture. Yet, many existing travel recommendation systems fall short, offering surface-level suggestions that miss the richness of the places they cover.

Growing up in Nepal, I've been immersed in its vibrant cultures, deep-rooted traditions, and diverse landscapes. This personal connection inspired me to create a travel assistant, powered by a retrieval-augmented generation (RAG) system, tailored specifically to Nepal. My life experience allows me to evaluate the system's cultural depth and accuracy with a level of insight that generic tools often lack.

1.2 Problem Statement

Most digital travel guides and recommendation tools today still follow a one-size-fits-all approach. They lean heavily on fixed itineraries and generic tourist info, leaving little room for personal interests or meaningful cultural context. As a result, travelers are left with recommendations that overlook their unique preferences, which don't thoughtfully reflect local culture, and can't answer the kinds of nuanced, real-world questions people often have when exploring a new place.

1.3 Proposed Solution

To bridge these gaps, I built a Retrieval-Augmented Generation (RAG) system that brings together several open-source Large Language Models (LLMs). The goal was to create a travel assistant that offers not just personalized and accurate advice, but also culturally rich insights, specifically for those planning to explore Nepal. I worked with models like **FLAN-T5 Large**, **FLAN-T5 Base**, and **TinyLLaMA**, and carefully compared how well each one handled complex, real-world travel questions.

This work was conducted as part of the DSCI 6004 course at the University of New Haven

1.4 Research Question

How well do different open-source LLMs **(FLAN-T5 Large, FLAN-T5 Base, and TinyLLaMA)** perform when it comes to accuracy, relevance, hallucination rate, and reasoning ability in the context of travel and cultural guidance?

2 RAG System and Implementation

2.1 System Overview

The RAG system I built is made up of three key parts: text embedding, information retrieval, and response generation using selected Large Language Models (LLMs).

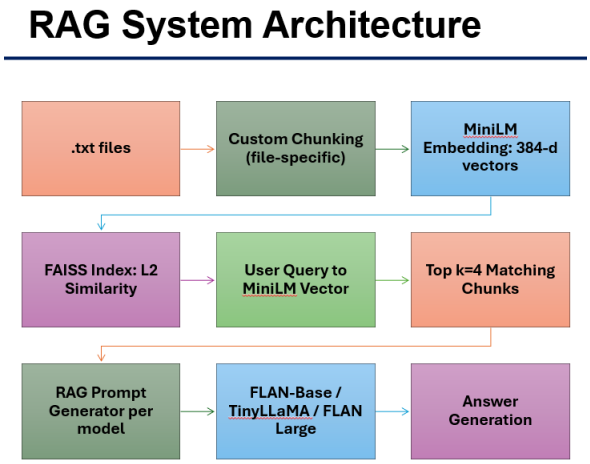

Figure 1: RAG system architecture for Nepal travel assistant.

To get started, I gathered a wide range of Nepal-specific travel content covering topics like accommodations, trekking routes, cultural festivals, visa policies, and local food from trusted online sources.

This content was then broken down into smaller, manageable chunks to improve retrieval performance. Each chunk was converted into a numerical format (known as an embedding) using the **MiniLM-L6-v2** Sentence Transformer. These embeddings capture the semantic meaning of the text and were indexed using FAISS (Facebook AI Similarity Search), which enables the system to quickly find the most relevant pieces of information in response to a user's query. These retrieved chunks then serve as the context for the selected LLMs to generate complete, context-aware answers tailored to travelers' questions.

2.2 Data Sources & Preprocessing

To build a solid dataset tailored to Nepal travel, I pulled information from a variety of high-quality sources. These included expert travel blogs, Nepal-focused tourism websites, the U.S. Department of State (for reliable visa information), and well-established commercial platforms like Hotels.com and Google Flights (for pricing data). This diverse mix helped ensure that the dataset covered both the practical details and the cultural nuances important to travelers.

When preprocessing the data, I used a targeted chunking strategy based on the type of content:

Food and Festivals: Divided at logical breakpoints such as specific dishes or individual festivals to keep each chunk self-contained and informative.

Accommodation and Pricing: Organized by price range and hotel rating to support quick, relevant lookups.

Trekking Guides (e.g., Everest Base Camp): Split into short, meaningful paragraphs to preserve detail and usability.

All the data was stored as plain .txt files without extra metadata, keeping the system fast and simple. In total, I ended up with 104 text chunks, each carefully curated to answer real-world questions a traveler might ask.

2.3 Embedding & Retrieval Details

I chose the **MiniLM-L6-v2** Sentence Transformer for generating text embeddings because of its efficiency, compact vector size **(384 dimensions),** and solid performance in semantic search tasks. This was especially important given the hardware limitations I faced—frequent RAM issues and package conflicts made heavier models impractical.

For retrieval, **FAISS** was the clear choice. It's fast, scalable, and designed to handle approximate nearest neighbor searches efficiently for delivering real-time results. Together, MiniLM and FAISS allowed the system to strike a strong balance between speed and relevance, making sure users get accurate and culturally informed answers from the LLMs.

3 Experiment Setup

3.1 Domain-Specific Questions

To thoroughly test how well the RAG-based travel assistant performs in a real-world setting, I crafted 13 questions that reflect what travelers commonly ask, ranging from logistics to cultural curiosity. These questions were designed to cover a broad range of topics, including food, accommodation, transportation, festivals, visas, and trekking. Here's a sample of the questions used in the evaluation:

- What is the must-try traditional food tourists should sample while visiting Nepal?
- What is the visa process and fee for U.S. citizens traveling to Nepal?
- What is the average round-trip flight cost from the United States to Kathmandu?
- Which major festivals should travelers plan around when visiting Nepal?
- What is the cheapest hotel or accommodation tourists can find in Nepal?

These questions served as a benchmark to test how well each model handled factual accuracy, relevance, reasoning, and how prone it was to hallucinating information.

3.2 Model Details

To fairly evaluate response quality, I selected three open-source, instruction-tuned LLMs. The choices were guided by a balance of capability, instruction-following skill, and compatibility with limited hardware resources (given repeated GPU memory crashes during development). Here's a quick rundown of the models:

1. FLAN-T5 Large and FLAN-T5 Base (by Google)

- **Architecture:** Encoder-decoder Transformer
- **Training:** Instruction-tuned for prompt-based tasks
- **Parameters:**
- **FLAN-T5 Large:** ~780 million
- **FLAN-T5 Base:** ~250 million
- **Token Limit:** 512 tokens (inputs were truncated when necessary)

Prompting Strategy:
Each prompt framed the model as a Nepal travel expert to provide context. Inputs were carefully trimmed to fit within token limits to avoid errors during indexing. These models are known for their solid instruction-following abilities, offering a good balance between performance and efficiency.

2. TinyLLaMA

- **Architecture:** Decoder-only Transformer (auto-regressive model)
- **Training:** Instruction-tuned for fast, lightweight inference
- **Parameters:** ~1.1 billion
- **Token Limit:** 1024 tokens (but inputs were capped at 350 tokens for consistency across models)

Prompting Strategy:
Like with FLAN-T5, prompts were structured with clear role instructions (e.g., "You are a Nepal travel expert...") and explicit context cues. The decoder-only nature of TinyLLaMA made it fast and efficient—ideal for low-resource environments—while still being capable of generating coherent, task-aware responses.

All models were evaluated using the same retrieval-augmented context, with an identical chunk-retrieval strategy to ensure fair and consistent comparisons across the board.

4 Results

4.1 Quantitative Comparison

Each model was evaluated on how well it responded to the 13 domain-specific travel questions, using a scoring scale from 0 (irrelevant or incorrect) to 5 (highly relevant and accurate). The table below summarizes the average scores:

Average Scores (13 Questions)

Model	Avg. Score (out of 5)	Observations
TinyLLaMA	3.77	High relevance, strong reasoning, but verbose and hallucination-prone
FLAN-T5 Large	2.23	Decent factual answers after prompt fix, but often too short or incomplete
FLAN-T5 Base	1.85	Very brief answers, limited reasoning, rarely complete

Figure 2: Model Accuracy Score

TinyLLaMA came out on top, significantly outperforming the FLAN-T5 variants. Its higher average suggests it was more effective in delivering relevant, detailed travel information within the RAG setup.

4.2 Qualitative Analysis

Beyond the numbers, a closer look at the responses revealed each model's unique strengths and limitations:

FLAN-T5 Large

- **Strengths:** Generally accurate on factual queries, especially those with straightforward answers like hotel prices or flight costs.
- **Weaknesses:** Responses were often too brief, lacking depth or cultural nuance. It occasionally hallucinated details when asked more specific or open-ended questions.
- **Example:** In a question regarding essentials for trekking query, it just generated the line saying "Trekking in Nepal guide" which shows the failure to answer open-ended questions.

FLAN-T5 Base

- **Strengths:** Fast and stable, with fewer hallucinations and minimal token-related errors.
- **Weaknesses:** Its answers were too generic to be useful for more nuanced questions, often failing to elaborate or provide specifics.
- **Example:** Similar to Flan-T5 large, it failed to answer the open-ended questions.

TinyLLaMA

- **Strengths:** Delivered rich, engaging responses with impressive fluency. It often captured the cultural and contextual layers of travel queries.
- **Weaknesses:** Prone to hallucination and repetition. It sometimes went overboard with overly long answers or repeated items.
- **Example:** For a food-related question, TinyLLaMA listed a wide variety of traditional dishes with detailed descriptions, but also repeated items and occasionally added unrelated content.

To summarize, while TinyLLaMA was the most detailed and engaging, it also carried a risk of hallucinations. FLAN-T5 Large offered a good balance for simple factual queries, and FLAN-T5 Base worked well for fast but shallow responses.

5 Discussion

Each model brought a different balance of strengths and weaknesses to the table. The key trade-offs observed are outlined below:

- **Accuracy vs. Fluency:** TinyLLaMA offered natural, fluent responses but sometimes sacrificed accuracy. FLAN-T5 models, particularly the large variant, favored correctness but were more terse.
- **Detail vs. Hallucination:** Richer, more engaging answers came at a cost. TinyLLaMA's strength in storytelling and elaboration also made it more likely to invent or repeat information.
- **Speed vs. Depth:** FLAN-T5 Base delivered factual answers quickly but lacked the depth needed for detailed planning. In contrast, TinyLLaMA was slower but better suited for exploratory or nuanced queries.

Ultimately, choosing the right model for a RAG-based travel assistant depends on the context. If speed and numeric precision are priorities, FLAN-T5 models—especially Large—are reliable. For more engaging, culturally aware guidance, TinyLLaMA stands out, as long as hallucination risks are monitored or mitigated.

6 Conclusion & Future Work

6.1 Summary of Main Findings

This study explored the capabilities of three open-source large language models—FLAN-T5 Large, FLAN-T5 Base, and TinyLLaMA—within a Retrieval-Augmented Generation (RAG) framework designed to deliver personalized travel

guidance and cultural insights about Nepal. Among the models evaluated, TinyLLaMA emerged as the most effective, achieving the highest average relevance score (3.77 out of 5). Its responses stood out for their richness, contextual awareness, and cultural depth.

However, this fluency came with trade-offs. TinyLLaMA was prone to hallucinations and often produced repetitive or overly verbose answers. FLAN-T5 Large, while generally accurate, tended to produce responses that were too concise to be useful for nuanced travel questions. FLAN-T5 Base delivered the fastest output but suffered from a lack of detail and specificity.

6.2 Recommended Improvements

To enhance the real-world utility of models like TinyLLaMA in a travel assistant context, several improvements are recommended:

- **Smarter Prompt Engineering:**

Structuring prompts more clearly and explicitly could help reduce hallucinations and repetitive phrasing.

- **Improved Context Retrieval:**

Using more sophisticated retrieval techniques, such as semantic reranking or adaptive chunking, could improve how relevant information is selected and integrated into responses.

- **Fact-checking Layers:**

Adding secondary validation steps, such as lightweight fact-checking models or external verification APIs, could help detect and filter out hallucinated content.

- **Hybrid Model Strategies:**

Combining models, using TinyLLaMA for fluency and a FLAN-T5 variant for factual verification, could balance depth and reliability more effectively.

- **Larger Models & More Data:**

Testing advanced models like fine-tuned versions of LLaMA or Mistral, trained specifically on travel-related content, could yield richer and more accurate responses.

- **Multilingual & Inclusive Expansion:**

Extending the system to support multiple languages and culturally diverse content would make it more accessible to international travelers and enhance its global relevance.

To conclude, this research highlights both the potential and the current limitations of open-source LLMs in specialized RAG systems. It lays the groundwork for smarter, more culturally aware travel assistants—and points toward meaningful next steps for scaling and refining such tools within the tourism domain.

References

[1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* In *Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*. https://dl.acm.org/doi/abs/10.5555/3495724.3496517

[2] Gautier Izacard and Edouard Grave. 2021. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online. Association for Computational Linguistics.

[3] Nils Reimers and Iryna Gurevych. 2019. Sentence BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Appendices

Appendix A: Complete List of Domain-Specific Questions

1. What is the must-try traditional food that tourists should eat while visiting Nepal?
2. How much do hotels typically cost in Kathmandu?
3. What is the visa process and fee for U.S. citizens traveling to Nepal?
4. What is the average roundtrip flight cost from the United States to Kathmandu?
5. Which major festivals should travelers plan around when visiting Nepal?

6. What are the best local transportation options for getting around Nepal as a tourist?
7. What is the recommended 10-day itinerary for trekking to Everest Base Camp?
8. What are the top 3 must-visit destinations in Nepal for first-time travelers?
9. Which are the top 3 most popular trekking routes in Nepal?
10. Which treks in Nepal are best for beginners?
11. Which treks are considered the most difficult for experienced trekkers?
12. What should tourists pack when preparing for a trek in the Himalayas?
13. What is the cheapest hotel or accommodation tourists can find in Nepal?

Appendix B: Dataset Description and Sources

Filename	Content Description	Sources
top9food.txt	Traditional foods and must-try dishes	Nepali Food Blog
Accomodation Info.txt	Hotel/accommodation prices and options in Nepal	Hotels.com, Booking.com
Entry Exit and Visa for US Citizen.txt	Visa procedures, fees for US travelers	US Embassy in Nepal
Flight Cost from USA Info.txt	Average flight prices from USA to Kathmandu	Google Flights
festivals in Nepal.txt	Major festivals celebrated in Nepal	Nepal Tourism Board, VisitNepal.com
Travel and Transportation.txt	Transportation methods within Nepal	Nepal Travel Guide
everestbasecamptrekguide.txt	Detailed Everest Base Camp Trek guide	Nepal Hiking Team
thingstodo_ktm.txt	Major tourist attractions and sightseeing in Kathmandu	TripAdvisor Kathmandu
trekking.txt	Trekking routes (beginner to advanced)	Asian Trekking

Appendix C: Chunking Logic Summary

Each dataset file was split into manageable text chunks using either regular expressions or fixed-length slicing based on its structure:

top9food.txt – Split by numbered list using regex: e.g., "1. Dal bhat, the national dish..."

Accomodation Info.txt – Fixed 500-character chunks: e.g., hotel names and pricing info.

Entry Exit and Visa for US Citizen.txt – Fixed 500-character chunks: e.g., visa rules for U.S. citizens.

Flight Cost from USA Info.txt – 400-character chunks: e.g., sample flight pricing from Google Flights.

festivals in Nepal.txt – Regex based on capitalized festival names: e.g., "Dashain\nThe biggest festival in Nepal...".

Travel and Transportation.txt – 400-character chunks: e.g., transportation types like taxis, buses.

everestbasecamptrekguide.txt – 500-character chunks: e.g., "Day 1: Fly Kathmandu to Lukla...".

thingstodo_ktm.txt – Regex based on city markers like "Durbar Square, Kathmandu".

trekking.txt – Regex by numbered titles (e.g., "1. Annapurna Base Camp Trek...").

This custom chunking ensured semantically meaningful segments were fed into the RAG system for retrieval.

Appendix D: Embedding and FAISS Index Details

Embedding Code (MiniLM-L6-v2):

```
from sentence_transformers import SentenceTransformer
import faiss

embedder = SentenceTransformer('all-MiniLM-L6-v2')
embeddings = embedder.encode(chunks).astype("float32")
print("Embedding shape:", embeddings.shape)  # (104, 384)

# Build FAISS index
index = faiss.IndexFlatL2(embeddings.shape[1])
index.add(embeddings)

# Save index
faiss.write_index(index, "nepal_rag_index.faiss")
```

Appendix E: RAG Response Function

```
def generate_rag_response(query, embedder, index, chunks, generator, model_type="flan", k=4, max_cha
    # Step 1: Retrieve top-k chunks
    top_chunks = search_faiss_index(query, embedder, index, chunks, k)

    # Limit context to avoid token overflow
    context = "\n".join(top_chunks)

    if model_type.startswith("flan"):
        context = context[:1400]

        # Step 2: Create safe prompt for FLAN models
        prompt = f"""Context:
{context}

{query}

Answer:"""

        # Step 3: Truncate by token length to max 512 using tokenizer
        inputs = flan_tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)
        prompt_trimmed = flan_tokenizer.decode(inputs["input_ids"][0], skip_special_tokens=True)

        result = generator(prompt_trimmed.strip(), max_new_tokens=300)

    elif model_type == "tinyllama":
    prompt = f"""You are a Nepal travel expert. Use the context below to answer the question briefly

Context:
{context}

Question:
{query}

Answer:"""
    result = generator(prompt.strip(), max_new_tokens=150)


    else:
        raise ValueError(f"Unsupported model_type: {model_type}")

    return result[0]['generated_text'].strip()
```

Appendix F: Scoring Rubric and Evaluation Examples

Scoring Rubric (Per Question, Scale: 0–5)

Score	Criteria
5	Fully accurate, highly relevant, well-structured, and reasoned.
4	Mostly accurate and relevant, minor factual/gap issues.
3	Partially correct, lacks detail, or has mild hallucinations.
2	Poor factual alignment, vague or irrelevant response.
1	Major hallucinations or unrelated content.
0	Blank, nonsensical, or completely wrong.

Scoring was based on relevance to the original dataset, factual correctness, reasoning clarity, and structure of the response.