Kaan Kosesoy

Room Search Documentation

System Architecture & Flow

This system helps users find hotel room images based on the features they describe in plain text. At the center of everything is a single class: RoomSearch. This class is responsible for all major tasks in the system — including managing data, calculating similarities, and working with the assistant agent.

Main Design: One Class to Handle Everything

All the logic and features of the system are managed through the RoomSearch class.

- Only one instance of this class is created.
- It takes two inputs when initialized:
    1. A list of hotel room image URLs
    2. A list of user queries (default search examples)

As soon as the object is created, it automatically calculates similarity scores between the default queries and the room images using three different methods. These results are then saved to files to avoid repeating the same work later.

Initialization Task: Preparing Everything in Advance

When the RoomSearch object is initialized, it runs an async background task that does the following:

1. Three Types of Similarity Matching

To match each image with the user queries, the system uses three different techniques:

- OpenAI GPT-4o (Direct Comparison)

Sends both the image and the query to the model, which gives a similarity score between 0 (worst match) and 1 (best match).

- TF-IDF Keyword Matching

First, it converts the image into a text description (using GPT-4o), then compares the keywords between that text and the user query.

- Semantic Embedding (Deep Learning)

Converts both the image description and the query into vector representations (using a sentence transformer), then compares them using cosine similarity.

2. Why Precompute Everything?

This setup needs a lot of API calls: downloading images, converting images to text, and running similarity comparisons for each image-query pair.

Even though some parts can be batched, image-based API calls can’t. So this quickly adds up to a high number of calls, which are:

- Slow (because of OpenAI rate limits)
- Expensive (because each call costs money)
- Error-prone (failures require retries)

To deal with this:

- The system calculates all similarities once at startup (takes 7–8 minutes).
- All results are saved to files, so they don't need to be recalculated later.
- If something fails halfway, it will resume from where it left off the next time it runs.
- During actual use (when the user talks to the assistant), results for the default queries are shown instantly.

This approach is much faster and more reliable than doing all calculations during user interaction.

Talking to the Assistant: LangGraph + RoomSearch

The system includes a smart assistant agent that talks with the user, understands what kind of room they want, and then asks the RoomSearch engine to find the best matching images. This is done using LangGraph/LangChain.

The Graph Has Three Nodes:

1. <u>Agent Node</u>
Talks to the user, understands their query, and cleans it up to make it suitable for searching.

2. <u>Tool Node</u>
Runs the actual search using the RoomSearch object.

3. <u>User Control Node</u>
Lets the user approve the query, choose the similarity method, or cancel the operation before running the tool.

The RoomSearch object is passed into the graph, so the agent and tools can use all of its features, like similarity scoring, image descriptions, and cached data.

Summary

- The RoomSearch class manages all parts of the system — image processing, similarity matching, and agent integration.

- During startup, it calculates and saves similarity scores for default queries so the system is ready to respond quickly during user interaction.

- The assistant talks to the user, cleans their input, and uses the system's resources to return the best matching hotel rooms.

- By caching everything and only calculating once, the system is faster, cheaper, and more robust.