* + .

**Mandatory Scorecards / Inputs**

1. **Customer Segment**
   * **Source**: User input on the frontend (e.g., family, business, solo traveler).
   * **Role**: Filters hotels that best match the customer’s segment.
   * **Normalization**: Not numeric; categorical filter.
2. **Location** (State / City)
   * **Source**: User input.
   * **Role**: Narrows down hotels to the selected state/city.
   * **Distance scoring (optional)**: If the user provides an exact address, **geopy** calculates distances and ranks closer hotels higher.
3. **Sentiment Analysis (AI Score)**
   * **Source**: Computed from hotel review embeddings using either:
     + **LLM-based method**: Pass embeddings or text to the model to get a score (0–1).
     + **Transformers sentiment model**: distilbert-base-uncased-finetuned-sst-2-english.
   * **Role**: Primary ranking metric reflecting review positivity.
   * **Normalization**: Already 0–1; averaged with normalized hotel rating.

**Optional / Filterable Scorecards**

1. **Price Range**
   * **Source**: Hotel metadata (price) + user-selected range.
   * **Role**: Filters hotels within budget; optional ranking if multiple hotels in range.
2. **Hotel Rating**
   * **Source**: Metadata from multiple booking sites (Booking.com, TripAdvisor, etc.).
   * **Role**: Combined with AI Score for sorting or used as filter (e.g., 3-star, 4-star).
   * **Normalization**: 0–1 scale for averaging with sentiment score.
3. **Address / Distance**
   * **Source**: Metadata (address) + geopy coordinates.
   * **Role**: Optional sorting if user wants hotels near a specific landmark.
4. **RAG-based Chatbot Access**
   * **Source**: FAISS vectorstore + embeddings + LLM.
   * **Role**: Allows users to query hotel facilities, reviews, or features via chatbot; answers come only from database.
   * **Effect on ranking**:Ranks according to the user preference..

**Corrected Summary Table**

| **Feature / Scorecard** | **Source** | **Used For** | **Normalization** |
| --- | --- | --- | --- |
| Customer Segment | Frontend input | Filter hotels | N/A |
| Location (State/City) | Frontend input / metadata | Filter hotels | N/A |
| Sentiment Analysis (AI Score) | Review embeddings + LLM / Transformers | Primary ranking | 0–1 |
| Price Range | Metadata / User input | optionalFilter | 0–1 |
| Hotel Rating | Metadata (multiple sites) | Optional filter | 0–1 |
| Address / Distance | Metadata + geopy | Optional proximity ranking | 0–1 |
| RAG-based Chatbot | FAISS + embeddings + LLM | User query only | N/A |

Why(explain the problem)

 Travelers find it difficult to select suitable hotels due to information overload and a lack of personalized recommendations.

 An AI-driven system is needed to provide tailored hotel suggestions.

 The system should base its recommendations on multiple features such as price, facilities, ratings, and customer preferences.

Explaining the methods(how)

### Data Preprocessing

The process begins with the raw data for each hotel, which includes reviews, ratings, facilities, and other details. This data is structured into a format suitable for the vectorization and analysis pipeline.

* **Review Text Cleaning:** Raw review text is cleaned to remove irrelevant characters, special symbols, and stop words. This step ensures that the sentiment analysis model focuses on meaningful words.
* **Metadata Extraction:** Structured data such as hotel name, city, state, price, and average ratings from various booking sites are extracted and stored as metadata associated with each hotel's review document. This metadata is crucial for subsequent filtering operations.

### Embeddings and Vector Storage

* The core of the system relies on embeddings, which are numerical representations of the hotel data.
* **Embedding Model:** The all-MiniLM-L6-v2 model is used to create these embeddings. This model is selected for its efficiency and ability to generate high-quality embeddings from text.
* **Vectorization:** The preprocessed review text and facility summaries for each hotel are fed into the all-MiniLM-L6-v2 model. The model converts this text content into a dense vector (a list of numbers) that semantically represents the hotel's reviews and amenities.
* **Vector Storage:** The generated vectors, along with their associated metadata (hotel name, city, price, etc.), are stored in a **FAISS vector store**. FAISS is chosen for its speed in performing similarity searches on large datasets of vectors, which is essential for quickly finding relevant hotels.

### Fetching and Ranking Hotels (Initial Recommendations)

* **Initial Filtering:** The user's input of customer\_segment, state, and city is used to perform an initial filter on the FAISS vector store's metadata. The system retrieves all hotel documents that match the selected city.
* **Sentiment Analysis:** The review text for each of the filtered hotels is then passed to the **distilbert-base-uncased-finetuned-sst-2-english** model. This model outputs a sentiment score (between 0 and 1) that quantifies the positivity of the reviews.
* **Rating Normalization:** The average\_rating from the hotel's metadata is normalized to a 0-1 scale.
* **AI Score Calculation:** The final "AI Score" for each hotel is calculated as the average of its sentiment score and its normalized average rating.
* **Ranking:** The hotels are then sorted in descending order based on this calculated AI Score. The system returns the top 5 hotels with their associated metadata and scores. If fewer than 5 hotels meet the criteria, it returns all of them.

### Filters

* Once the initial top 5 hotels are displayed, users can apply optional filters to refine their search. These filters work by operating on the hotels that are already in the system's database.
* **Price Range:** The system filters for hotels that fall within the user-specified price range. The resulting subset is then re-sorted by the AI Score to maintain relevance.
* **Hotel Star Rating:** Users can select a star rating (e.g., 3-star, 4-star) to filter the results. The remaining hotels are then ranked again by their AI Score.
* **Address/Location:** This is a two-step process:
  1. The system first attempts **fuzzy matching** to find hotels with addresses close to the user's input.
  2. If fuzzy matching fails, the system uses **geopy** to get the latitude and longitude of the user's input and calculates the distance to each hotel. It then returns the 5 nearest hotels, ranked by their AI Score.

### Summary Generation with LLM

* The system also uses an LLM to provide human-readable summaries for hotel reviews and facilities.
* **Review Summary:** The embeddings for each hotel's reviews are passed to the LLM (meta-llama/llama-4-scout-17b-16e-instruct). The LLM processes this information and generates a concise summary of the key points from the reviews, highlighting what guests liked or disliked.
* **Hotel Facilities Summary:** Similarly, the embeddings containing information about hotel facilities are passed to the same LLM. The LLM then generates a clean, readable summary of the available facilities and amenities.
* **RAG-based Chatbot:** The same LLM is used to power the RAG chatbot. When a user asks a question about a hotel's facilities, the system retrieves the relevant embeddings from the FAISS database and uses the LLM to generate an answer based *only* on the retrieved information. This ensures the chatbot's responses are accurate and non-hallucinatory.

What (explain the solution)

### The Solution: An AI-Driven Hotel Recommendation System

The solution is an **AI-driven hotel recommendation system** that provides tailored suggestions for travelers. It addresses the problem of information overload by synthesizing both quantitative and qualitative data into a single, comprehensive score. The system is built with a **React frontend** and a **FastAPI backend** for a modern, scalable, and responsive user experience.

#### Core Features:

* **Hybrid Ranking System (AI Score)**: The system’s core innovation is the **"AI Score"**, which is used to rank hotels. This score is calculated by taking the average of two key metrics:
  1. A **sentiment score** derived from an analysis of the hotel’s reviews.
  2. A **normalized average rating** from various booking sites. This approach provides a more holistic and accurate ranking than traditional methods that rely solely on star ratings.
* **Guided User Flow**: The application guides the user through a series of selections, starting with their **customer segment** (e.g., business traveler, family, solo traveler), then **state**, and finally **city**. This progressive filtering process quickly narrows down the initial list of potential hotels.
* **Dynamic Filtering**: After the initial recommendations are displayed, users can apply optional filters to refine their search. These include:
  1. **Price range**: Users can specify a minimum and maximum price, and the system will filter hotels accordingly.
  2. **Hotel star rating**: Users can filter hotels by their star rating (e.g., 3-star, 4-star).
  3. **Location**: For a more granular search, the system uses **geopy** and **fuzzy matching** to find hotels near a specific address, returning the nearest options.
* **Interactive UI**: The frontend displays the recommended hotels with rich details to help users make an informed decision. For each hotel, it shows the **AI Score**, **average rating**, **individual ratings from other booking sites**, a **summary of hotel facilities**, and a **review summary** generated by an LLM. Clicking on a hotel image redirects the user to the booking page via the pageurl. The UI also features buttons for filters and a **Google Map** that displays all five recommended hotels using location data from SerpAPI.
* **RAG-Based Chatbot**: An integrated chatbot allows users to ask questions about specific hotel facilities or amenities. The chatbot uses **Retrieval-Augmented Generation (RAG)**, which means it retrieves relevant information from the database and uses an LLM to generate an answer based **only** on that retrieved information. This prevents the chatbot from hallucinating or providing inaccurate information.

### The Workflow in Detail:

1. **Landing Page & Customer Flow**
   * **Landing Page**: The user's initial entry point, with a "Next" button to begin.
   * **Customer Segment Selection**: The user selects a customer segment, which can be used to inform later recommendations.
   * **State and City Selection**: The user narrows down their search by first selecting a state and then a specific city.
2. **Hotel Filtering & Initial Recommendation**
   * The backend (FastAPI) filters all hotel documents in the FAISS vector store based on the chosen city.
   * The review text for each filtered hotel is passed through the distilbert-base-uncased-finetuned-sst-2-english model to generate a sentiment score.
   * The hotel’s average rating is normalized to a 0-1 scale.
   * The **AI Score** is calculated as the average of the sentiment score and the normalized rating.
   * The top 5 hotels with the highest AI Score are returned to the frontend.
3. **Optional Filters**
   * Users can apply filters for price, star rating, and location to dynamically refine the displayed hotels. Within each filtered subset, hotels are always re-sorted by their AI Score.
4. **Data Display on Frontend**
   * For each hotel, the frontend displays its **AI Score**, **average rating**, **individual booking site ratings**, a **summary of hotel facilities**, and a **review summary**. Hotel images are fetched via **SerpAPI**, and a **Google Map** at the top shows the locations of all five hotels.
5. **RAG Chatbot**
   * The chatbot uses the **FAISS vector store** and the **meta-llama/llama-4-scout-17b-16e-instruct LLM** to provide accurate answers to user questions about hotel facilities, only using data from its database.

### The Solution: An AI-Driven Hotel Recommendation System

The solution is an **AI-driven hotel recommendation system** that provides tailored suggestions for travelers. It addresses the problem of information overload by synthesizing both quantitative and qualitative data into a single, comprehensive score. The system is built with a **React frontend** and a **FastAPI backend** for a modern, scalable, and responsive user experience.

#### Core Features:

* **Hybrid Ranking System (AI Score)**: The system’s core innovation is the **"AI Score"**, which is used to rank hotels. This score is calculated by taking the average of two key metrics:
  1. A **sentiment score** derived from an analysis of the hotel’s reviews using the distilbert-base-uncased-finetuned-sst-2-english model.
  2. A **normalized average rating** from various booking sites. This approach provides a more holistic and accurate ranking than traditional methods that rely solely on star ratings.
* **Guided User Flow**: The application guides the user through a series of selections, starting with their **customer segment** (e.g., business traveler, family, solo traveler), then **state**, and finally **city**. This progressive filtering process quickly narrows down the initial list of potential hotels.
* **Dynamic Filtering**: After the initial recommendations are displayed, users can apply optional filters to refine their search. These include:
  1. **Price range**: Users can specify a minimum and maximum price, and the system will filter hotels accordingly.
  2. **Hotel star rating**: Users can filter hotels by their star rating (e.g., 3-star, 4-star).
  3. **Location**: For a more granular search, the system uses **geopy** and **fuzzy matching** to find hotels near a specific address, returning the nearest options.
* **Interactive UI**: The frontend displays the recommended hotels with rich details to help users make an informed decision. For each hotel, it shows the **AI Score**, **average rating**, **individual ratings from other booking sites**, a **summary of hotel facilities**, and a **review summary** generated by an LLM. Clicking on a hotel image redirects the user to the booking page via the pageurl. The UI also features buttons for filters and a **Google Map** that displays all five recommended hotels using location data from SerpAPI.
* **RAG-Based Chatbot**: An integrated chatbot allows users to ask questions about specific hotel facilities or amenities. The chatbot uses **Retrieval-Augmented Generation (RAG)**, which means it retrieves relevant information from the database and uses an LLM to generate an answer based **only** on that retrieved information. This prevents the chatbot from hallucinating or providing inaccurate information.

### The Workflow in Detail:

1. **Landing Page & Customer Flow**
   * **Landing Page**: The user's initial entry point, with a "Next" button to begin.
   * **Customer Segment Selection**: The user selects a customer segment, which can be used to inform later recommendations.
   * **State and City Selection**: The user narrows down their search by first selecting a state and then a specific city.
2. **Hotel Filtering & Initial Recommendation**
   * The backend (FastAPI) filters all hotel documents in the FAISS vector store based on the chosen city.
   * The review text for each filtered hotel is passed through the distilbert-base-uncased-finetuned-sst-2-english model to generate a sentiment score.
   * The hotel’s average rating is normalized to a 0-1 scale.
   * The **AI Score** is calculated as the average of the sentiment score and the normalized rating.
   * The top 5 hotels with the highest AI Score are returned to the frontend.
3. **Optional Filters**
   * Users can apply filters for price, star rating, and location to dynamically refine the displayed hotels. Within each filtered subset, hotels are always re-sorted by their AI Score.
4. **Data Display on Frontend**
   * For each hotel, the frontend displays its **AI Score**, **average rating**, **individual booking site ratings**, a **summary of hotel facilities**, and a **review summary**. Hotel images are fetched via **SerpAPI**, and a **Google Map** at the top shows the locations of all five hotels.
5. **RAG Chatbot**
   * The chatbot uses the **FAISS vector store** and the **meta-llama/llama-4-scout-17b-16e-instruct LLM** to provide accurate answers to user questions about hotel facilities, only using data from its database.

### Why use FastAPI, All-MiniLM, and DistilBERT?

* **FastAPI** is a high-performance Python web framework ideal for building APIs. It is known for its speed, automatic documentation, and support for asynchronous programming, which allows it to handle many concurrent requests efficiently.
* **All-MiniLM-L6-v2** is a highly efficient transformer model that generates **384-dimensional dense vectors** from text, capturing the semantic meaning of sentences and paragraphs. It is designed for tasks like semantic search and clustering, making it perfect for creating our hotel embeddings.
* **DistilBERT** is a smaller, faster, and more efficient version of BERT. It retains about 97% of BERT's performance while being 40% smaller and 60% faster, making it an excellent choice for tasks like sentiment analysis where quick responses are crucial.

Benefits and architecture

### Solution Benefits

* **Improved Accuracy and Trustworthiness:** The system combines numerical ratings with a sentiment score from reviews, creating a holistic "AI Score". This provides a more nuanced ranking than traditional star ratings, building user trust.
* **Reduced Hallucinations:** The **RAG chatbot** is a key benefit as it prevents the LLM from generating inaccurate or fabricated information. It grounds its responses in the verified, internal hotel database, ensuring answers about facilities and amenities are accurate and reliable.
* **Enhanced User Productivity:** The streamlined workflow, dynamic filters, and on-demand chatbot help users quickly find the perfect hotel, reducing the time and effort spent on searching.
* **Scalability and Performance:** The architecture is designed for high performance and scalability, making it suitable for handling a large volume of users and data. Using **FastAPI** with its asynchronous capabilities allows it to manage multiple requests concurrently, ensuring low latency and efficient resource utilization.
* **Cost-Effectiveness**: By using the lightweight **DistilBERT** model for sentiment analysis, the system gains a significant performance boost while being more computationally efficient and requiring less memory than larger models like BERT.

### Architecture

The system is built on a robust, multi-component architecture that leverages modern AI and web development technologies.

#### Frontend (React)

The user interface is built with **React**, which provides a dynamic and interactive experience. It handles:

* The multi-page user flow for selecting the customer segment, state, and city.
* Displaying the top 5 recommended hotels with all their details (images, AI score, summaries, etc.).
* User interactions with the optional filters and the chatbot interface.
* Rendering the Google Map that shows the locations of all recommended hotels.

#### Backend (FastAPI)

The **FastAPI** backend is the central hub of the system. It's built on a Python framework that is high-performance and well-suited for building APIs for a recommendation engine. Key functions include:

* **API Endpoints:** It provides endpoints for each step of the user flow, from selecting a city to applying filters and interacting with the chatbot.
* **Data Validation:** It uses Python's type hints to automatically validate incoming data, which helps prevent errors and ensures a reliable data flow.
* **Dependency Injection:** Its powerful dependency injection system simplifies the management of complex dependencies like the vector store and external API connections.
* **Asynchronous Processing:** FastAPI's native support for asynchronous programming allows it to handle multiple concurrent requests, which is crucial for real-time applications like a recommendation system.

#### Data Storage and AI Models

* **FAISS Vector Store:** The system uses **FAISS** (Facebook AI Similarity Search) to store the hotel embeddings. FAISS is an optimized library for performing fast similarity searches on large datasets of vectors. This allows the backend to quickly retrieve relevant hotel information based on user queries and filters.
* **Embedding Model:** The **all-MiniLM-L6-v2** model is used to convert hotel review and facility texts into a **384-dimensional dense vector space**. This model is compact, fast, and accurate for creating embeddings for semantic search and clustering.
* **Sentiment Analysis Model:** The **distilbert-base-uncased-finetuned-sst-2-english** model is a smaller, more efficient version of BERT. It is used to perform sentiment analysis on hotel reviews, providing a score between 0 and 1.
* **LLM (for RAG and Summaries):** The **meta-llama/llama-4-scout-17b-16e-instruct LLM** is used for two main tasks: generating concise review and facility summaries from the embeddings, and powering the RAG chatbot to answer specific user questions.
* **External APIs:** **SerpAPI** is used to fetch hotel images and location data, which are then displayed on the frontend and the Google Map.

#### Deployment

The entire system will be deployed on **AWS**, with the FastAPI backend likely running on **AWS EC2** or within **Docker containers** on **AWS ECS** to ensure scalability and reliability.

### Challenges and Innovations

Building a robust hotel recommendation system comes with a unique set of challenges, from data acquisition to model implementation. Our project has innovatively addressed these by moving beyond traditional methods.

#### 1. Dataset Acquisition

* **Challenge:** Finding a comprehensive, clean, and well-structured dataset with all the necessary columns was a significant hurdle. Publicly available datasets are often limited and may lack key attributes such as geolocation information, detailed hotel facilities, or a sufficient number of reviews. A major challenge we faced was that major booking sites hold their data very privately. Many of them have aggressive anti-scraping measures, and they have strict privacy policies that prohibit the use of their data for third-party applications. Additionally, their official APIs are typically expensive, with costs that are prohibitive for a hackathon project.
* **Innovation:** We had to curate our own dataset, collecting heterogeneous data from multiple sources to obtain a complete set of attributes. This involved combining numerical ratings with unstructured review text and other crucial metadata like city, address, and pageurl. We also manually assigned unique IDs to each hotel to ensure data integrity.

#### 2. Sentiment Analysis

* **Challenge:** Our initial thought was to use traditional sentiment analysis, which focuses on simple polarity (positive/negative) and subjectivity. While straightforward, this approach provides a limited view of user opinions, failing to capture the nuances of hotel reviews.
* **Innovation:** We chose a more advanced method by using a fine-tuned transformer model, **distilbert-base-uncased-finetuned-sst-2-english**, for sentiment analysis. This model is more efficient and accurate at capturing the sentiment of unstructured text. By using this model, we generate a sentiment score between 0 and 1, which provides a more granular measure of a review's positivity. This score is then averaged with the normalized average rating to create our **"AI Score,"** a key innovation that allows for a more comprehensive ranking of hotels.

#### 3. Location-Based Filtering

* **Challenge:** Our initial idea for location-based filtering was to use simple keyword matching (e.g., searching for "hotel near park street" by looking for the keywords "park street"). This approach proved to be highly inaccurate and unreliable due to variations in naming conventions, spelling mistakes, and the lack of a proper geographical context.
* **Innovation:** We upgraded our approach to a two-step process that combines **fuzzy matching** and **geopy**. Fuzzy matching is used first to find a close match for the address within our database. If that fails, we use the geopy library to get the exact latitude and longitude of the user's query. The system then calculates the distance between the query location and each hotel's coordinates, returning the top 5 nearest hotels. This method provides a much more accurate and robust solution for location-based recommendations.

Uniqueness

The uniqueness of this hackathon project lies in its innovative approach to hotel recommendations, which combines multiple data sources and advanced AI techniques to provide a highly personalized and reliable user experience.

### Uniqueness of the AI Score

The most unique aspect of the system is the creation of the **"AI Score."** Traditional recommendation systems often rely on a single, aggregated star rating, which can be misleading. Our system goes beyond this by creating a composite score that incorporates both the quantitative average rating from different booking sites and a qualitative sentiment score from customer reviews.

* **Granular Sentiment Analysis:** Instead of just a simple positive or negative label, we use a fine-tuned transformer model, **distilbert-base-uncased-finetuned-sst-2-english**, to generate a sentiment score between 0 and 1. This gives a more granular measure of how positive the reviews are, allowing us to accurately rank hotels with similar star ratings based on the actual guest experience. This approach allows for a more nuanced understanding of the reviews, which can be highly subjective. The final AI Score is a simple average of the sentiment score and the normalized average rating, giving equal weight to both guest sentiment and numerical performance.

### Uniqueness of Location-Based Filtering

Our solution for location-based filtering is unique because it combines different techniques to provide a highly accurate and flexible search.

* **Beyond Keyword Matching:** Our initial approach of simple keyword matching proved unreliable due to typos and different ways of writing addresses.
* **Fuzzy Matching & Geopy:** We innovated by implementing a two-step process. We first use **fuzzy matching** to find similar addresses even with misspellings or variations. If this fails, we use **geopy** to get the precise geographic coordinates of the user's input and calculate the distance to each hotel. This ensures that the system can accurately find the nearest hotels, providing a robust solution that goes beyond basic filtering.

### Uniqueness of the RAG Chatbot

The integration of a **RAG-based chatbot** is a significant differentiator that sets our project apart.

* **Guarding Against Hallucinations:** Most chatbots are prone to "hallucinations" (generating inaccurate or misleading information) when they lack specific knowledge. Our RAG chatbot is unique because it's trained to only answer questions using information retrieved from its internal database (the FAISS vector store).
* **Contextual and Accurate Answers:** When a user asks a question about hotel facilities, the chatbot retrieves relevant embeddings and uses the LLM to generate a response that is **grounded in the factual data**. This ensures that every answer is accurate and directly relevant to the specific hotel, building user trust and providing a reliable source of information.

### ****Quantified Benefits of Our AI-Driven Hotel Recommendation System****

1. **Improved Guest Experience**
   * Personalized recommendations using **AI Score** (sentiment + ratings) rank hotels based on real guest satisfaction.
   * Estimated impact: **50–70% faster hotel selection** and higher booking confidence, leading to **increased guest loyalty**.
2. **Operational Efficiency**
   * Automates routine tasks such as analyzing reviews, ranking hotels, and responding to user queries via **RAG chatbot**.
   * Reduces staff workload by **30–50%**, freeing time for strategic and guest-focused tasks.
3. **Revenue Optimization**
   * Dynamic ranking and sentiment-aware insights allow hotels to **capitalize on high-demand periods** and offer personalized upsells.
   * Potential revenue increase: **10–20% more bookings per high-demand period**.
4. **Enhanced Personalization**
   * Provides review summaries, hotel facilities, and AI-driven insights for each user segment.
   * Increases user engagement and satisfaction; users see **top hotels that truly match their preferences** rather than generic listings.
5. **Faster, Smarter Decision-Making**
   * Combines **sentiment analysis, hotel ratings, price range, location filters, and AI insights** to produce **top 5 hotel recommendations instantly**.
   * Users no longer need to manually compare dozens of hotels — saving time and improving decision quality.

### ****Consequences If These Benefits Are Missing****

1. **Loss of Competitive Advantage**
   * Without AI-based personalization and sentiment insights, the system becomes just another generic booking tool.
   * Lower engagement as users turn to competitors with smarter recommendations.
2. **Inefficient Operations**
   * Manual processing of reviews, rankings, and queries increases staff workload and risk of human error.
   * Critical tasks like inventory management and pricing adjustments take longer.
3. **Reduced Revenue Opportunities**
   * Without AI-driven recommendations and dynamic insights, hotels may miss bookings and upsell opportunities.
   * High-demand periods might be underutilized due to slow response.
4. **Limited User Satisfaction**
   * Users are forced to rely on static ratings and textual reviews.
   * Lack of AI insights, RAG chatbot, and sentiment scoring results in a **less personalized and slower hotel selection process**.
5. **Lower Decision Accuracy**
   * Without combining multiple scorecards (sentiment, hotel rating, price, location, customer segment), recommended hotels may **not truly reflect user preferences**.
   * Leads to decreased trust in the system and lower overall satisfaction.

### Timelines: Time to Realize Benefits

The benefits of our project can be realized in a short period of time, as is typical for hackathon initiatives. Our timeline for realizing these benefits is phased, starting from the hackathon's inception on August 29, 2025.

### Immediate Benefits (Within 1 week)

* **Proof of Concept:** The most immediate benefit is the creation of a working proof of concept. By the end of the hackathon, we will have a functional prototype that demonstrates the core features, such as the AI Score and the RAG-based chatbot. This validates the project's feasibility and its unique value proposition.
* **Data Acquisition:** The challenge of data acquisition is handled prior to the hackathon. For the event, the dataset is **static**, which allows the team to focus on building the core logic and features without the complexities of real-time data ingestion.
* **Rapid Innovation:** The competitive and time-bound nature of the hackathon allows for accelerated product development and the rapid validation of new ideas without the extended timelines of traditional development cycles.
* **Skill Enhancement:** The team will immediately realize the benefit of learning and applying new technologies such as FastAPI, FAISS, and specific AI models, which are valuable skills for the future.

### Short-Term Benefits (1-3 months)

* **User Feedback and Iteration:** After the hackathon, we can begin gathering user feedback to refine the system. This iterative process will allow us to improve the accuracy of our recommendations and the overall user experience.
* **Scaling the System:** We will begin the process of deploying the system on **AWS**, setting up a scalable backend and optimizing API calls to handle a larger number of users. This is when the project will transition from a prototype to a tangible, scalable product.

### Mid-Term Benefits (3-6 months)

* **Revenue Generation:** Once the system is live and stable, we can begin to realize the benefit of revenue through affiliate commissions and, eventually, B2B licensing. This is the phase where the business model begins to take shape.
* **Data-Driven Improvements:** As more users interact with the system, we will be able to use the collected data to continuously improve our models and algorithms. This continuous learning will enhance the accuracy and relevance of our recommendations.

### Long-Term Benefits (6+ months)

* **Market Share and Brand Recognition:** A successful and well-regarded application will help us gain market share and build a reputation as an innovative provider of travel technology. This will position us to compete with established players in the travel industry.
* **Attracting Talent:** The project can be used as a portfolio piece to attract top talent and create a recruiting pipeline for tech-savvy individuals who want to work on innovative solutions.

Investments: aws 1 year free trial . serpapi:250 requests a month

Team name: GROUP 2 (same for all)

College name: HERITAGE INSTITUTE OF TECHNOLOGY(same for all)

Team Menber name: TIASHA MAITRA, MD.FAIZAN AKHTAR, SHAMIK BHATTACHARYA, BITTU MITRI MODAK, ANKITA NATH, RISHI KARMAKAR, AYUSH GUHA, PUSHKAR RAJ

Mail id: tiasha.maitra.aiml26@heritageit.edu.in, [md.faizanakhtar.aiml26@heritageit.edu.in](mailto:md.faizanakhtar.aiml26@heritageit.edu.in), shamik.bhattacharya.aiml26@heritageit.edu.in, [bittu.mitrimodak.aiml26@heritageit.edu.in](mailto:bittu.mitrimodak.aiml26@heritageit.edu.in), [ankita.nath.cse26@heritageit.edu.in](mailto:ankita.nath.cse26@heritageit.edu.in), [rishi.karmakar.aiml26@heritageit.edu.in](mailto:rishi.karmakar.aiml26@heritageit.edu.in),

[ayush.guha.aiml26@heritageit.edu.in](mailto:ayush.guha.aiml26@heritageit.edu.in), [pushkar.raj.aiml26@heritageit.edu.in](mailto:pushkar.raj.aiml26@heritageit.edu.in)

department : CSE(AIML) same for all

Year of passing: 2026( same for all)