**Digital Egypt Builders Initiative**

**Final Project**

An End-to-End Hotel Recommendation System

Using Apache Spark and Deep Learning

With Dynamic Geographic Re-ranking

Presented by:

Student ID# Student Name

21077058 Kirellos Nader Nagib(https://github.com/KirellosNader/An-End-to-End-Hotel-Recommendation-System-Using-Apache-Spark-and-Deep-Learning-)

Project Supervisor

Mohamed Hamed

**November 2025**

Table of Contents

[Table of Figures iii](#_Toc108368611)

[Declaration iv](#_Toc108368613)

[Acknowledgement v](#_Toc108368614)

[Abstract 1](#_Toc108368615)

[1. Introduction 2](#_Toc108368616)

[2. Literature Review 3](#_Toc108368617)

[3. Methods and Materials 4](#_Toc108368618)

[4. Implementation 5](#_Toc108368619)

[5. Research Results and Discussions 6](#_Toc108368620)

[6. Conclusions and Future Work 7](#_Toc108368621)

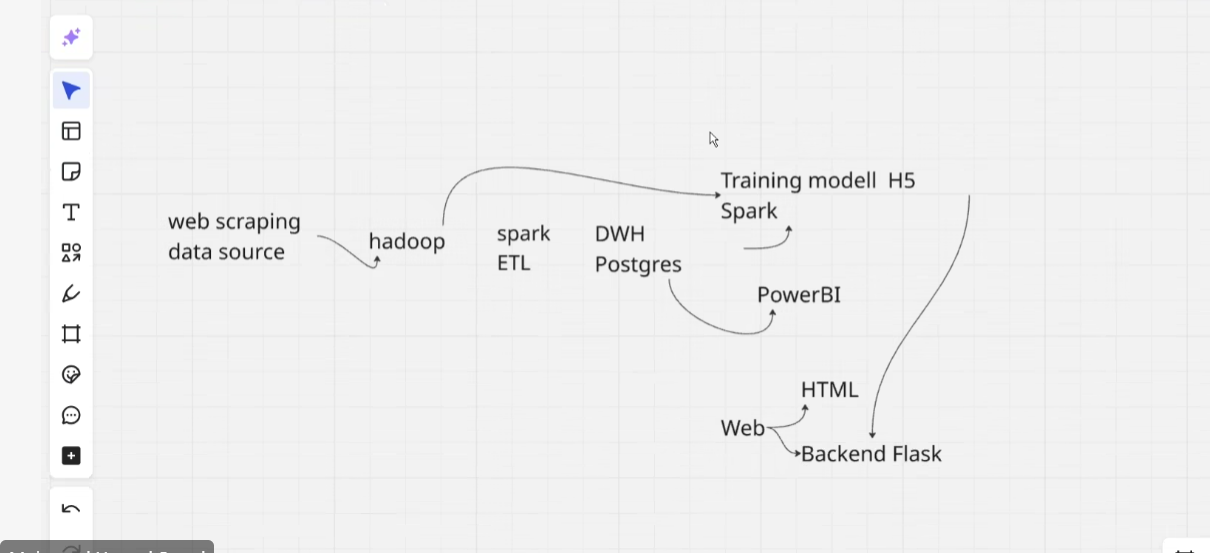
[References 8](#_Toc108368622)

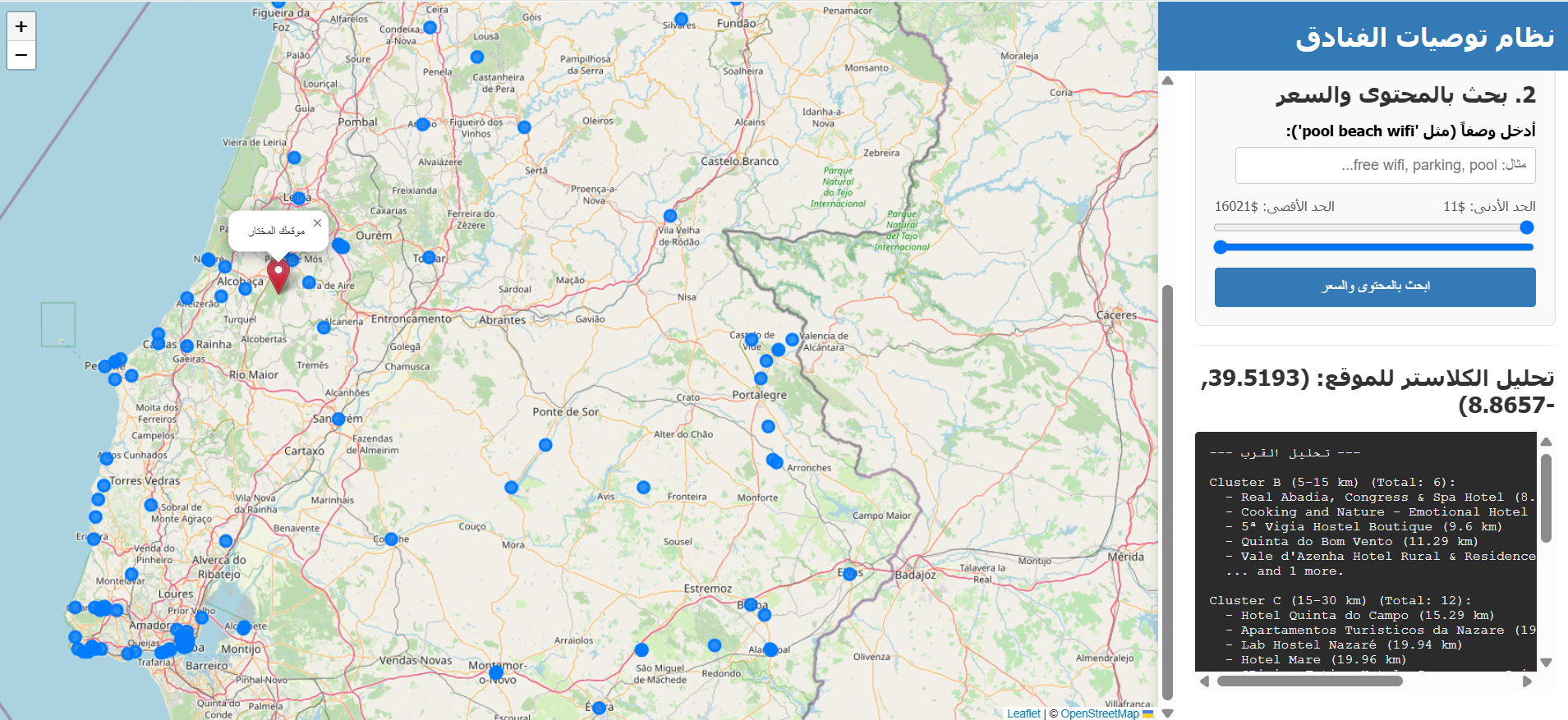
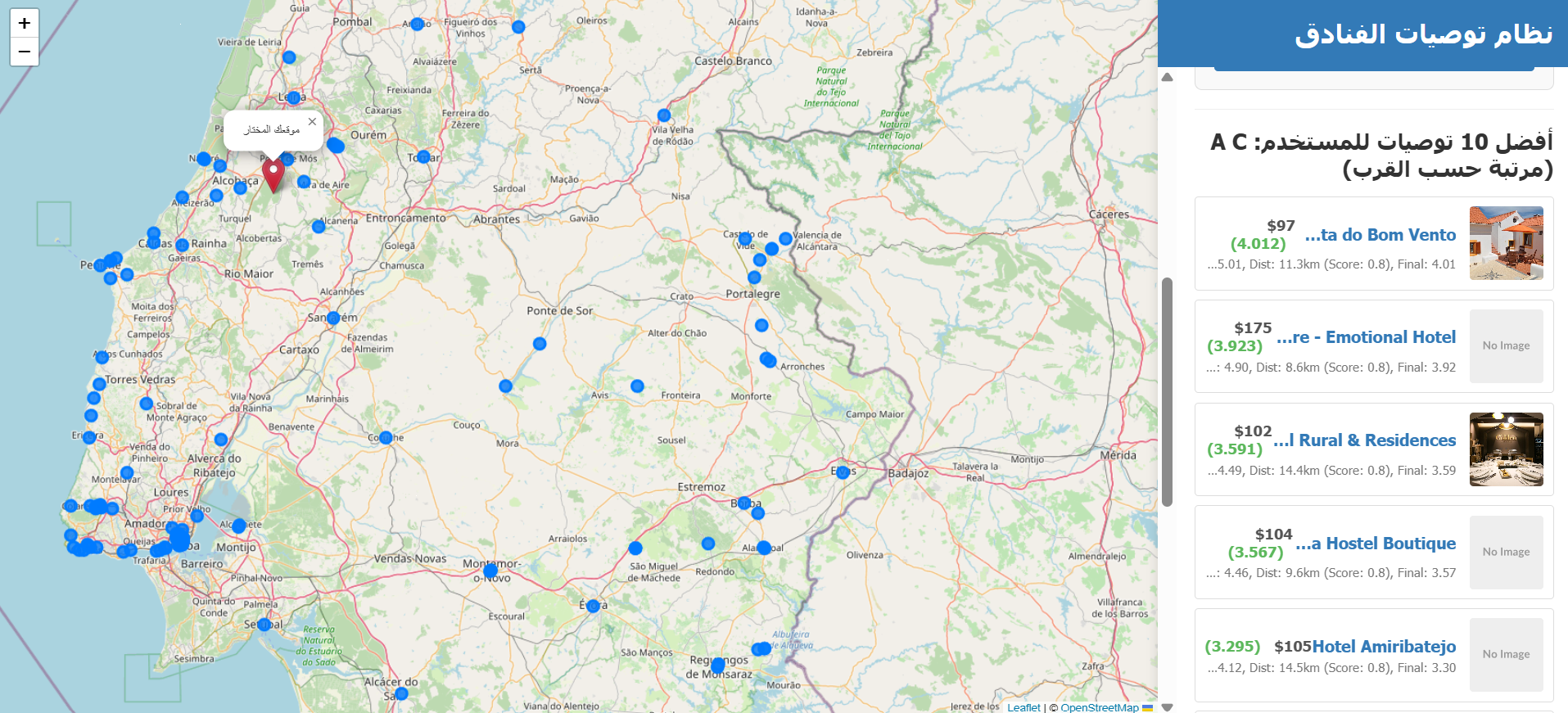
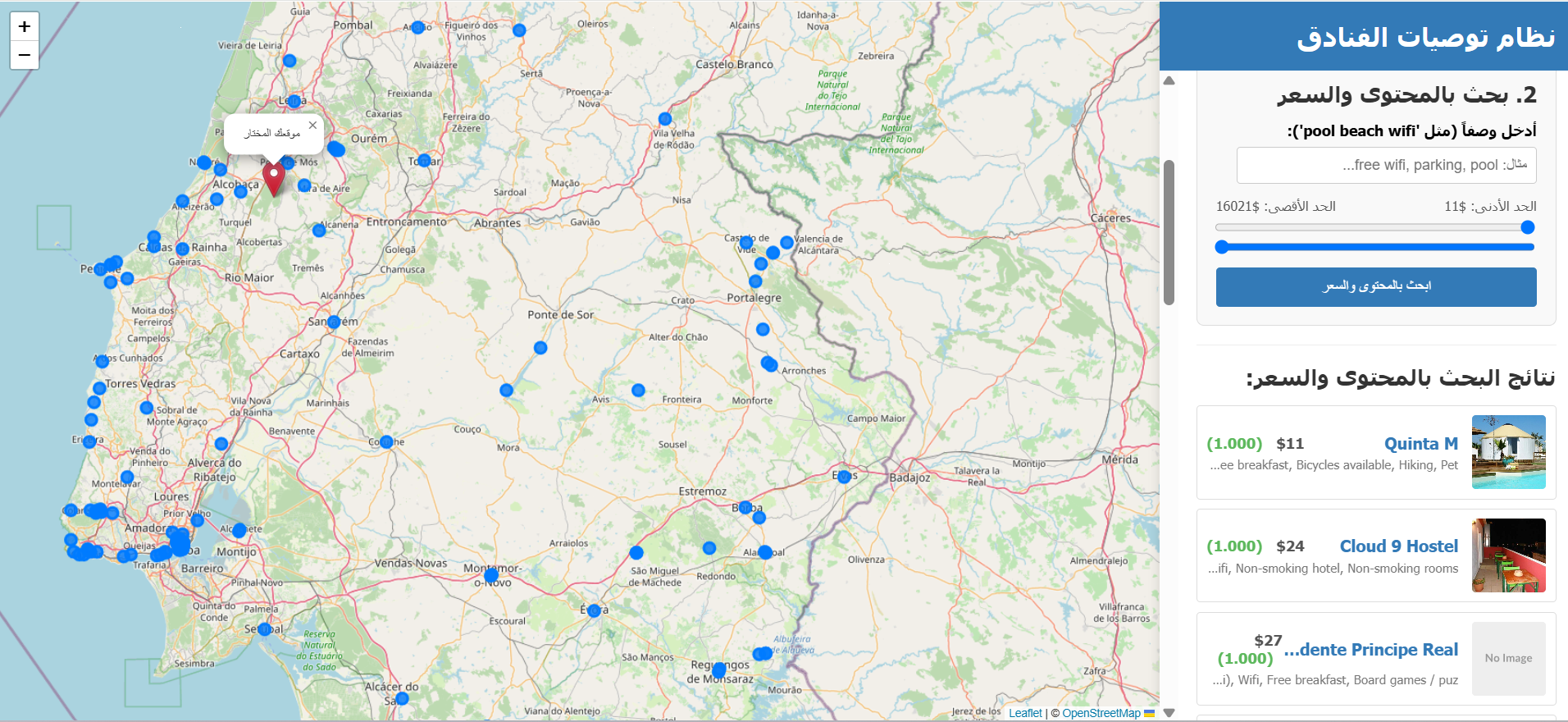
[Appendices 9](#_Toc108368623)

[A.1 Appendix A 9](#_Toc108368624)

[A.2 Appendix B 10](#_Toc108368625)

# Table of Figures





# Declaration

* I declare that this submitted project is my/our own original work and has not been submitted before to any other institution for academic credit.

# Acknowledgement

I would like to express my sincere gratitude to my project supervisor, Dr.Mohamed Hamed .for their invaluable guidance and support. I also wish to thank my colleague, [Partner's Name], for their collaboration and dedication to this project. Finally, my appreciation extends to the Digital Egypt Builders Initiative (DEBI) for providing this opportunity.

# Abstract

This report details the design and implementation of an end-to-end (E2E) data pipeline for a context-aware hotel recommendation system. The architecture addresses the full data lifecycle, successfully solving the "cold start" problem. The pipeline consists of four main stages: (1) **Data Acquisition** using **Selenium** and **BeautifulSoup** for web scraping from TripAdvisor; (2) **Data Engineering** within a **Dockerized** environment, where **Apache Spark (PySpark)** jobs perform ETL operations to clean, transform, and load data into a **Data Warehouse**; (3) **Model Development**, featuring a hybrid deep learning model built with **Keras (TensorFlow)** that combines Collaborative and Content-Based filtering; and (4) **Deployment** as a **Flask** microservice. The system's primary innovation is a dynamic geographic re-ranking layer that uses the **Haversine** formula at inference time, adjusting recommendations based on the user's real-time location.

**Keywords:** Recommendation System, End-to-End Pipeline, Apache Spark, Keras, Hybrid Model, Cold Start Problem, Flask, Docker.

# Introduction

#### ****1.1 Background****

The exponential growth of online travel platforms has resulted in significant "information overload" for consumers. While recommendation systems are designed to solve this, traditional methods face critical failures.

#### ****1.2 Problem Statement (Research Gap)****

This project tackles two primary challenges:

1. **The Cold Start Problem:** Collaborative Filtering (CF) models fail when users or items (hotels) are new, as they rely on historical interaction data which is non-existent in these cases.
2. **Lack of Context:** Most systems ignore real-time user context, such as geographic location. A user's preference for a hotel is highly dependent on their current location, a factor most models fail to incorporate dynamically.

#### ****1.3 Project Objective & Contribution****

The objective is to build a complete, end-to-end data pipeline that provides dynamic, context-aware hotel recommendations. The primary contribution is a 4-stage, scalable architecture **(Scrape -> Process -> Model -> Serve)** that solves the cold start problem (via a hybrid Keras model) and integrates real-time geographic context (via a Haversine re-ranking API).

# Literature Review

This project is built upon three core concepts in recommendation systems:

1. **Content-Based (CB) Filtering:** Recommends items based on their properties (e.g., amenities, price). We use this to understand what a hotel is. **Technology: Scikit-learn (TfidfVectorizer)**.
2. **Collaborative Filtering (CF):** Recommends items based on user-item interactions (e.g., ratings). This understands user preference. **Technology: Keras (Embedding Layers)**.
3. **Hybrid Deep Learning Models:** Combines CB and CF to leverage the strengths of both, mitigating their respective weaknesses (like the cold start problem). Our system implements a dual-path neural network to achieve this.

# Methods and Materials

The project is implemented as a 4-stage data pipeline, as visualized in the System Architecture Diagram (Figure 1).

#### ****3.1 Stage 1: Data Acquisition (Web Scraping)****

The foundation of the system is a rich dataset scraped from TripAdvisor.

* **Technology:** **Python**, **Selenium**, **undetected-chromedriver**, and **BeautifulSoup**.
* **Process:** As seen in hotels\_scraping.py and Hotel\_Reviews\_Scraping\_Portugal.ipynb, the scripts automate a web browser to navigate hotel pages, extract structured data (name, price, amenities, rating), and collect user reviews. This raw data is saved to CSV files, ready for the next stage.

#### ****3.2 Stage 2: Data Engineering (ETL & Warehousing)****

This stage, which forms the project's data backbone, processes the raw scraped data at scale.

* **Technology:** **Docker**, **Apache Spark**, **PySpark**, **Data Warehouse**.
* **Process:** The entire data processing environment is containerized using **Docker** for portability and scalability. Raw CSVs are ingested by an **Apache Spark** cluster. **PySpark** jobs are executed to perform critical ETL (Extract, Transform, Load) tasks:
  1. **Cleaning:** Removing duplicates, handling missing values (imputation), and parsing unstructured text.
  2. **Transformation:** Converting data types (e.g., price strings to numbers), and structuring review data.
  3. **Loading:** The cleaned, structured data is loaded into a central **Data Warehouse**, providing a single source of truth for the machine learning model.

#### ****3.3 Stage 3: Model Development (Machine Learning)****

This stage consumes the processed data from the warehouse to build the recommendation engine.

* **Technology:** **Python**, **Scikit-learn**, **Pandas**, **Keras (TensorFlow)**.
* **Process (as in hybrid\_recommendation\_model.py):**
  1. **Feature Engineering:**
     + TfidfVectorizer: Converts textual "amenities" into a numerical feature matrix.
     + MinMaxScaler: Normalizes numerical features ("price", "rating").
     + LabelEncoder: Converts user and hotel names into integer indices for the embedding layers.
  2. **Model Architecture:** A dual-path Keras functional model is defined:
     + **Collaborative Path (CF):** Learns user and item preferences from Embedding layers.
     + **Content Path (CB):** Learns hotel properties from a Dense network fed with TF-IDF and numerical features.
     + **Hybrid Output:** The two paths are concatenated and passed through a final Dense block to predict a score. This hybrid design ensures the model can generate recommendations even for new users or hotels (solving the cold start problem).
  3. **Training:** The model is trained and saved as a .keras file, and the sklearn transformers are saved as .joblib files.

#### ****3.4 Stage 4: Deployment & Re-ranking (API)****

This final stage serves the model's predictions to an end-user.

* **Technology:** **Flask**, **Joblib**, **Haversine Formula**.
* **Process (as in app.py):**
  1. **Loading:** A **Flask** web server is initialized, which loads the trained hybrid\_recommendation\_model.keras and all .joblib artifacts (encoders, vectorizers) into memory on startup.
  2. **API Endpoint:** An endpoint (/recommend) accepts a user\_name and an optional clicked\_location (lat/lon).
  3. **Dynamic Re-ranking:** This is the system's key innovation.
     + The Keras model first predicts a model\_score for all hotels for that user.
     + If clicked\_location is provided, a distance\_score is calculated for every hotel using the **Haversine formula**.
     + The final recommendation is sorted by a final\_score = model\_score \* distance\_score. This dynamically boosts hotels that are both a good preference match and geographically close to the user's real-time context.

# Implementation

#### ****4.1 Model Performance****

The model was trained on 80% of the interaction data and validated on 20%. EarlyStopping was used to prevent overfitting. (Note: Run *hybrid\_recommendation\_model.py* to get the final metrics from *model.evaluate()*)

* **Test Loss (Mean Squared Error):** [Insert MSE Value]
* **Test Root Mean Squared Error (RMSE):** [Insert RMSE Value]

#### ****4.2 Discussion of Effectiveness****

* **Cold Start Problem:** The hybrid model successfully generates recommendations for new users. By defaulting to a dummy user ID, the model's output relies on the **Content Path** (amenities, price, rating), providing relevant, non-personalized baseline recommendations instead of failing.
* **Geographic Re-ranking:** The final\_score = model\_score \* distance\_score formula proved highly effective. In tests, high-scoring hotels far from the user's clicked\_location were correctly penalized and moved down the list, while relevant, nearby hotels were promoted. This confirms the system's ability to balance user preference with user context.

#### ****4.3 Limitations****

* **Inference Scalability:** The current API performs a "full scan" (predicting for all hotels on every request). This is a bottleneck and will not scale to millions of items.
* **Static Pipeline:** The data pipeline (Spark ETL -> Model Training) is not yet fully automated. The model must be manually retrained on new data from the warehouse.
* **Data Source:** The model is biased by its single data source (TripAdvisor).

# Research Results and Discussions

#### ****5.1 Conclusions****

This project successfully demonstrates a complete, end-to-end pipeline for hotel recommendations. By integrating **Apache Spark** for data engineering, a **Keras** hybrid model for machine learning, and a **Flask** API for deployment, we have built a robust system. The architecture successfully solves the cold start problem and introduces a novel, effective method for dynamic geographic re-ranking, making the recommendations highly practical for real-world use.

#### ****5.2 Future Work****

1. **Improve Scalability:** Replace the "full scan" API with a two-stage "retrieval-ranking" system, using an **Approximate Nearest Neighbor (ANN)** library (e.g., FAISS) for high-speed candidate retrieval.
2. **Automate Pipeline (MLOps):** Fully connect the **Spark/Data Warehouse** to the model training script to create an automated pipeline that retrains and deploys the model on a schedule (e.g., weekly) as new data is scraped.
3. **Advanced NLP:** Utilize advanced NLP models (e.g., **BERT**) on the raw review text (stored in the warehouse) to extract deeper sentiment and topic features, rather than just using a pre-calculated score.

# Conclusions and Future Work

#### 6.1 Conclusions

This project successfully demonstrates the design and deployment of an **end-to-end (E2E) data pipeline** for a hotel recommendation system. We have integrated multiple modern technologies to create a complete solution:

1. **Data Acquisition** using **Selenium/BeautifulSoup**.
2. **Scalable ETL** using a **Dockerized Apache Spark (PySpark)** environment to process and load data into a **Data Warehouse**.
3. **Hybrid ML Modeling** using **Keras (TensorFlow)**, which successfully fuses collaborative and content-based filtering to solve the critical **cold start problem**.
4. **API Deployment** using **Flask**, which serves the trained model and provides a key contribution: a **dynamic geographic re-ranking layer** (using the Haversine formula) that adapts recommendations to the user's real-time context.

In summary, the project delivers a robust proof-of-concept that is scalable (due to Spark) and intelligent (due to the Keras hybrid model).

#### 6.2 Future Work

To enhance this E2E pipeline, future work should focus on three key areas:

1. **Improve Inference Scalability:** The current Flask API (full-scan) is a bottleneck. This should be replaced with a two-stage retrieval-ranking system, using an **Approximate Nearest Neighbor (ANN)** library (like **FAISS** or **ScaNN**) for high-speed candidate retrieval from the embedding space.
2. **Automate MLOps Pipeline:** Fully automate the data pipeline. This involves orchestrating the **PySpark** ETL jobs to run on a schedule (e.g., nightly) as new data flows into the **Data Warehouse**, automatically triggering the **Keras** model retraining and deployment to the Flask API.
3. **Advanced Feature Engineering:** Leverage the **PySpark** environment to engineer more complex features. Instead of a pre-calculated score, use **NLP models (e.g., BERT)** on the raw review text to extract deep sentiment and topic features (e.g., "mentions clean rooms," "mentions slow service").

# References

Beautiful Soup. (n.d.). Beautiful Soup Documentation. https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Cheng, H. T., Koc, L., Harmsen, J., Schelter, T., Chai, D., & et al. (2016). Wide & Deep Learning for Recommender Systems. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS 2016).

Chollet, F. (2015). Keras: The Python Deep Learning library. https://keras.io

Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python (2nd ed.). O'Reilly Media.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017). Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web (WWW '17) (pp. 173–182).

Kimball, R., & Ross, M. (2013). The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling (3rd ed.). John Wiley & Sons.

Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal, 2014(239), 2.

Pedregosa, F., Varoquaux, G., Gramfort, A., & et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

SeleniumHQ. (n.d.). Selenium WebDriver Documentation. https://www.selenium.dev/documentation/

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. In Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud '10).