Northwestern University
CS–310 Scalable Software Architecture: Final Project

Team: MLDS-Lambda-ANS
Hye Won Hwang – hwh0745
Alejandra Lelo de Larrea Ibarra – ali8110
Sharika Mahadevan – smv7369

Rental Wizard App

When you want to rent an apartment and place it on an apartment rental website like Zillow you have at least two problems: figure out the price at which you want to rent the apartment and write an eye-catching description. The **Rental Wizard App** will help customers to get a fair rental price and a captivating announcement leveraging machine learning (ML), Generative AI and AWS services.

As students of the MS in ML and Data Science, we were interested in developing a full ML pipeline with serverless architecture and to implement generative AI within the AWS Framework. Figure 1 shows the cloud architecture that supports this app. The client, developed in Streamlit, will be able two request 3 services: *train_pipeline, suggest_price, make_description.* Each service is called using AWS API Gateway (server) and executed by AWS Lambda functions. Note that the *train_pipeline* service will only be called by an employee of the company, while the *suggest_price* and *make_description* services will be called by our customers. For demonstration purposes and for the scope of this project, we added the three services in the same server and client, but for deployment they should be managed separately. We briefly describe each service next.

train_pipeline:
When clients invoke the *train_pipeline* service, they must specify whether they require new training or re-training of the data. For new training, clients provide the URL of the source data (such as the UCI Repository) and the name of the model-configuration file. This information triggers three sequential Lambda functions for data ingestion, data cleaning, and model training. For re-training, clients only submit the model configuration file name, prompting two sequential Lambda functions for data cleaning and model training.

Data ingestion is performed by a Lambda function that downloads raw data from the source and uploads it to the S3 bucket in the */data/raw* folder.

Data cleaning and feature engineering occur next, rectifying any discrepancies in the raw data and deriving new features or modifying existing ones, with the cleaned data saved in the */data/clean* folder.

The pipeline then trains a Random Forest Regressor model, using k-fold cross-validation for hyperparameter tuning. The artifacts generated (train and test set, cross-validation results, best model, scores, etc.) are uploaded to the S3 bucket under the */modeling_artifacts* folder. Note that the model configuration file, in YAML format, which allows different model configurations to be run (set of hyperparameters to test, set of features to use, etc), should be pre-uploaded to the S3 bucket in the */config* folder. It is important to emphasize that the project's focus is not on prediction accuracy but on implementing an end-to-end training pipeline in a serverless manner. For complex model training, the Lambda function may be replaced with a Docker containerized environment managed by AWS ECS (Elastic Container Service).

suggest_price:

The *suggest_price* service is triggered when a client makes a request through AWS API Gateway. This activates an AWS Lambda function that retrieves the stored model from the S3 bucket and predicts a fair rental price based on the client's input data about the property, which could include details like the number of bathrooms, bedrooms, square footage, any extra fees, and a list of amenities.

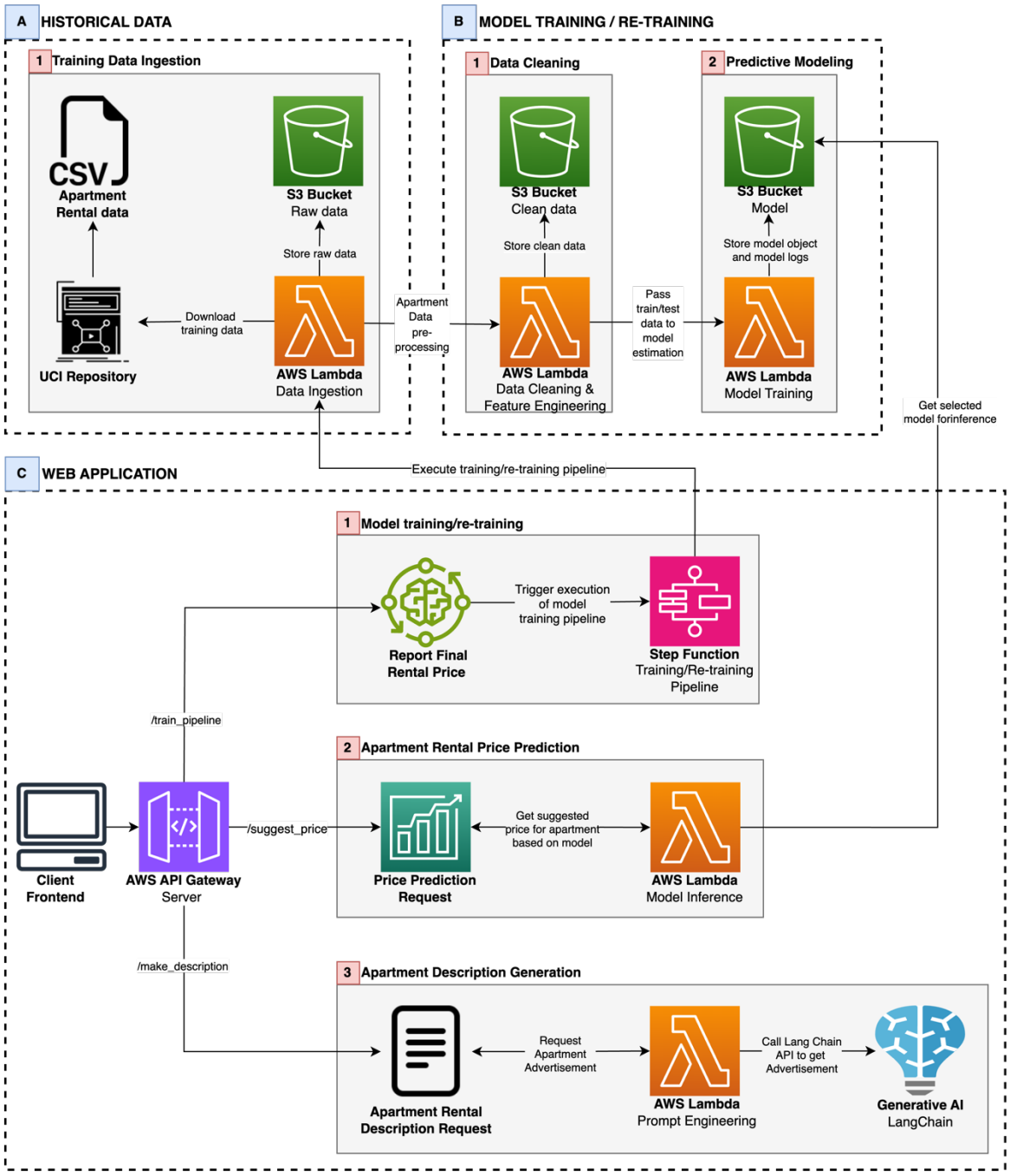
make_description:

Clients may also request the generation of an apartment description (advertisement) via the frontend. This service requires clients to input property details such as the number of bathrooms, bedrooms, square footage, extra fees, and amenities. Upon receiving a request, an AWS Lambda function is activated, prompting the Generative AI chain that leverages a Large Language Model (LLM) API to produce creative and appealing apartment descriptions.

Future implementation

When clients request services on our app, they are providing new data that could be used to fine tune the price prediction model. As a step to be implemented in the future, we would like to add a module that stores the apartment features that each client is providing and asks the clients to report the “final rental price” to be able to trigger an automated retraining of the current model when we reach X number of new observations. Moreover, we would like to implement a model drift detection so that, when our predictions are off by a certain threshold, the train pipeline is triggered again.

Figure1: Rental Wizard App Cloud Architecture



A1 **Raw Data Collection.** The raw data pertaining to apartment rentals is sourced directly from the UCI Repository. This data is automatically downloaded using an AWS Lambda function, which subsequently saves the collected data into an S3 bucket for further processing.

B1 **Data Cleaning & Feature Engineering.** Data cleaning and feature engineering is performed using AWS Lambda function to remove any discrepancies or errors, and derive new features or modify existing ones. Processed data is saved to S3 bucket.

B2 **Model Training.** A ML model is trained using the cleaned and feature-engineered data. The training process is serverless using AWS Lambda. The model metadata and final model object is stored into an S3 bucket. Note that, for more complex model training we might need to replace the Lambda function with a Docker containerized environment, managed by AWS ECS (Elastic Container Service).

C1 **Model Training/Retraining Service.** Client (company employee) requests the execution of the model training/re-training for predicting the rental price for apartments via a frontend interface. On receiving a request, an AWS Step-Function is executed to run the model training/re-training pipeline. If model training is triggered for the first time, the client has to pass the source url for the data repository and model-config file. If model is being re-trained, the step-function will skip the data ingestion and the client only passes the model-config file.

C2 **Price Prediction Service.** The client (customer) requests a rental price prediction via a frontend interface. The request triggers an AWS Lambda function designed for model inference. The function fetches the stored model from the S3 bucket and provides a price prediction based on the client's input data.

C3 **Apartment Description Generation Service.** Clients request the generation of an apartment description (advertisement) via the frontend. On receiving a request, an AWS Lambda function is activated to prompt the generation of an advertisement. This Lambda function interacts with a Generative AI chain, leveraging a Large Language Model (LLM) API to produce creative and appealing apartment descriptions.