**Step 1: Environment Setup**

***-> Install Python and Set Up Conda Environment***

First, ensure Python 3.9 is installed and then set up a virtual environment using Conda:

bash

# Install Anaconda if not already installed (follow the instructions on their official website)

conda create --name jerichoai\_env python=3.9

conda activate jerichoai\_env

***-> Install Required Libraries***

Install all necessary libraries for data handling, machine learning, and web development:

bash

conda install numpy pandas scikit-learn matplotlib seaborn

pip install tensorflow==2.6.0

conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch

pip install fastapi uvicorn

**Step 2: Data Handling**

***->Set up data manipulation using Pandas and NumPy:***

**p**ython

import numpy as np

import pandas as pd

# Example function to load and preprocess data

def load\_data(filepath):

data = pd.read\_csv(filepath)

data.fillna(0, inplace=True) # Handling missing values

return data

# Simulated data loading

# data = load\_data('path\_to\_cloud\_usage\_data.csv')

**Step 3: Model Development**

***->Develop cost prediction and anomaly detection models using TensorFlow and PyTorch:***

**->TensorFlow Model for Cost Prediction**

python

import tensorflow as tf

def build\_and\_train\_model(data):

model = tf.keras.Sequential([

tf.keras.layers.Dense(10, activation='relu', input\_shape=(data.shape[1],)),

tf.keras.layers.Dense(1)

])

model.compile(optimizer='adam', loss='mse')

model.fit(data, epochs=10, batch\_size=32)

return model

# model = build\_and\_train\_model(preprocessed\_data)

**-> PyTorch Model for Anomaly Detection**

python

import torch

import torch.nn as nn

class AnomalyDetector(nn.Module):

def \_\_init\_\_(self):

super(AnomalyDetector, self).\_\_init\_\_()

self.layer1 = nn.Linear(10, 64)

self.layer2 = nn.Linear(64, 1)

def forward(self, x):

x = torch.relu(self.layer1(x))

return self.layer2(x)

# Example usage might involve creating an instance of this model and training it with cloud usage data

**Step 4: API Development for Integration**

* Develop an API using FastAPI to serve the machine learning model predictions:

python

from fastapi import FastAPI

from pydantic import BaseModel

app = FastAPI()

class RequestData(BaseModel):

data: list # Example of data structure expected

@app.post("/predict")

def predict(request\_data: RequestData):

# Example function to process data and return predictions

# predictions = model.predict(request\_data.data)

return {"predictions": "simulated\_response"}

# Run using: uvicorn module\_name:app --reload

**Step 5: Version Control and CI/CD Setup**

* **Use Git for version control. Implement CI/CD pipelines using GitHub Actions for automated testing and deployment.**

yaml

# .github/workflows/python-app.yml

name: Python application

on:

push:

branches: [ main ]

pull\_request:

branches: [ main ]

jobs:

build:

runs-on: ubuntu-latest

steps:

**- uses: actions/checkout@v2**

**- name: Set up Python 3.9**

**uses: actions/setup-python@v2**

**with:**

**python-version: 3.9**

**- name: Install dependencies**

**run: |**

**python -m pip install --upgrade pip**

**pip install -r requirements.txt**

**- name: Test with pytest**

**run: |**

**pytest**

**Step 6: Deployment and Monitoring**

Deploy APIs using Docker and manage using Kubernetes. Monitor the application using Prometheus and Grafana.

Dockerfile

# Example Dockerfile

FROM python:3.9-slim

COPY . /app

WORKDIR /app

RUN pip install -r requirements.txt

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "80"]

**FINAL STEP FOR DATA GOVERNANVE : IMPLIMENTING JERICHO AI:**

def main():

print("Welcome to the Cloud Cost Comparison Tool!")

# Links to cloud billing pages

aws\_link = "https://aws.amazon.com/billing/"

gcp\_link = "https://console.cloud.google.com/billing/"

print("\nPlease visit the following links to access your billing information:")

print(f"AWS Billing: {aws\_link}")

print(f"GCP Billing: {gcp\_link}")

# Input number of instances for AWS and GCP

num\_aws = int(input("\nHow many AWS instances do you have? "))

num\_gcp = int(input("How many GCP instances do you have? "))

aws\_instances = []

gcp\_instances = []

# Collecting AWS instances data

for i in range(num\_aws):

name = input(f"Enter the name for AWS instance {i+1}: ")

cost = float(input(f"Enter the bill amount for AWS instance {name} ($): "))

aws\_instances.append({'name': name, 'cost': cost})

# Collecting GCP instances data

for i in range(num\_gcp):

name = input(f"Enter the name for GCP instance {i+1}: ")

cost = float(input(f"Enter the bill amount for GCP instance {name} ($): "))

gcp\_instances.append({'name': name, 'cost': cost})

# Process AWS instances

aws\_total = sum(instance['cost'] for instance in aws\_instances)

aws\_useless = [instance['name'] for instance in aws\_instances if instance['cost'] == 0]

# Process GCP instances

gcp\_total = sum(instance['cost'] for instance in gcp\_instances)

gcp\_useless = [instance['name'] for instance in gcp\_instances if instance['cost'] == 0]

# Results

print("\nAnalysis Results:")

if aws\_total < gcp\_total:

print("AWS is cheaper.")

elif gcp\_total < aws\_total:

print("GCP is cheaper.")

else:

print("Both AWS and GCP have the same total billing amount.")

# Advising on useless instances

if aws\_useless:

print(f"Consider removing the following AWS instances with zero-dollar billing as they are useless: {', '.join(aws\_useless)}")

if gcp\_useless:

print(f"Consider removing the following GCP instances with zero-dollar billing as they are useless: {', '.join(gcp\_useless)}")

# Output links again for user reference

print("\nFor more details on your billing, please revisit the following links:")

print(f"AWS Billing Details: {aws\_link}")

print(f"GCP Billing Details: {gcp\_link}")

if \_\_name\_\_ == "\_\_main\_\_":

main()