Open sidebar

ChatGPT o1-preview

ChatGPT o1-preview

Share User

You said:

I need help finishing Phase 2 CODE. MLFlow: file:///C:/foai/mlruns/0/c58955aa3bbf43a9b838abc4769280b2/artifacts/traffic_prediction_model/data/model.keras file:///C:/foai/mlruns/0/c58955aa3bbf43a9b838abc4769280b2/artifacts/traffic_prediction_model/data/keras_module.txt Path: file:///C:/foai/mlruns/0/c58955aa3bbf43a9b838abc4769280b2/artifacts/traffic_prediction_model/MLmodel artifact_path: traffic_prediction_model flavors: keras: data: data keras_backend: tensorflow keras_version: 3.5.0 save_exported_model: false python_function: env: conda: conda.yaml virtualenv: python_env.yaml loader_module: mlflow.keras python_version: 3.12.4 mlflow_version: 2.16.2 model_size_bytes: 1012493 model_uuid: 8b6eaabc2eed4779817eac7b2a8a0b34 run_id: c58955aa3bbf43a9b838abc4769280b2 utc_time_created: '2024-10-08 04:13:32.477059' Path: file:///C:/foai/mlruns/0/c58955aa3bbf43a9b838abc4769280b2/artifacts/traffic_prediction_model/conda.yaml channels: - conda-forge dependencies: - python=3.12.4 - pip<=24.2 - pip: - mlflow==2.16.2 - keras==3.5.0 - numpy==1.26.4 - tensorflow-intel==2.17.0 name: mlflow-env Path: file:///C:/foai/mlruns/0/c58955aa3bbf43a9b838abc4769280b2/artifacts/traffic_prediction_model/python_env.yaml python: 3.12.4 build_dependencies: - pip==24.2 - setuptools==72.1.0 - wheel==0.43.0 dependencies: - -r requirements.txt Path: file:///C:/foai/mlruns/0/c58955aa3bbf43a9b838abc4769280b2/artifacts/traffic_prediction_model/requirements.txt mlflow==2.16.2 keras==3.5.0 numpy==1.26.4 tensorflow-intel==2.17.0 2024-10-03 18:28:10,618 - Building LSTM model with input shape (10, 1) 2024-10-03 18:28:10,634 - Starting training for LSTM model... 2024-10-03 18:28:18,436 - Model training complete. MAE: 0.08178112004919455, MSE: 0.008384029096638166, R2: 0.9335198238521533 2024-10-03 18:28:41,358 - LSTM Model - MAE: 0.08178112004919455, MSE: 0.008384029096638166, R2 Score: 0.9335198238521533 2024-10-03 18:31:09,899 - Building LSTM model with input shape (10, 1) 2024-10-03 18:31:09,902 - Starting training for LSTM model... 2024-10-03 18:31:17,165 - Model training complete. MAE: 0.10006858775468808, MSE: 0.012571809122514278, R2: 0.9003133129276727 2024-10-03 18:31:31,950 - LSTM Model - MAE: 0.10006858775468808, MSE: 0.012571809122514278, R2 Score: 0.9003133129276727 2024-10-03 18:38:51,701 - Building LSTM model with input shape (10, 1) 2024-10-03 18:38:51,706 - Starting training for LSTM model... 2024-10-03 18:38:58,173 - Model training complete. MAE: 0.11519860795117379, MSE: 0.016379962779301482, R2: 0.8701170048062237 2024-10-03 18:39:18,223 - LSTM Model - MAE: 0.11519860795117379, MSE: 0.016379962779301482, R2 Score: 0.8701170048062237 2024-10-03 18:46:42,027 - Building LSTM model with input shape (10, 1) 2024-10-03 18:46:42,032 - Starting training for LSTM model... 2024-10-03 18:46:51,333 - Model training complete. MAE: 0.061447702620241344, MSE: 0.005071332717761161, R2: 0.9597874615539814 2024-10-03 18:47:06,985 - LSTM Model - MAE: 0.061447702620241344, MSE: 0.005071332717761161, R2 Score: 0.9597874615539814 2024-10-03 18:50:22,277 - Building LSTM model with input shape (10, 1) 2024-10-03 18:50:22,289 - Starting training for LSTM model... 2024-10-03 18:50:29,336 - Model training complete. MAE: 0.1003314728029851, MSE: 0.012557193916351803, R2: 0.9004292024920956 2024-10-03 18:50:44,426 - LSTM Model - MAE: 0.1003314728029851, MSE: 0.012557193916351803, R2 Score: 0.9004292024920956 2024-10-03 19:01:59,006 - Building LSTM model with input shape (10, 3) 2024-10-03 19:01:59,014 - Starting training for LSTM model... 2024-10-06 17:54:28,165 - Building LSTM model with input shape (10, 1) 2024-10-06 17:54:28,170 - Starting training for LSTM model... 2024-10-06 17:54:35,491 - Model training complete. MAE: 0.07879188189369488, MSE: 0.007809961578005144, R2: 0.9380718249627867 2024-10-06 17:54:54,609 - LSTM Model - MAE: 0.07879188189369488, MSE: 0.007809961578005144, R2 Score: 0.9380718249627867 2024-10-06 18:16:27,049 - Building LSTM model with input shape (10, 207) 2024-10-06 18:16:27,054 - Starting training for LSTM model... 2024-10-06 18:18:59,302 - Model training complete. MAE: 0.038002186919906, MSE: 0.00858810208558167, R2: 0.9177395006372965 2024-10-06 18:19:21,792 - LSTM Model - MAE: 0.038002186919906, MSE: 0.00858810208558167, R2 Score: 0.9177395006372965 2024-10-06 19:14:47,262 - Building LSTM model with input shape (10, 207) 2024-10-06 19:14:47,273 - Starting training for LSTM model... 2024-10-06 19:16:27,754 - Model training complete. MAE: 0.039729712449196174, MSE: 0.0091402932169781, R2: 0.9124503788080872 2024-10-06 19:16:27,899 - Model saved as lstm_model.pkl 2024-10-06 19:16:27,991 - LSTM Model - MAE: 0.039729712449196174, MSE: 0.0091402932169781, R2 Score: 0.9124503788080872 2024-10-08 00:04:55,564 - Building LSTM model with input shape (10, 207) 2024-10-08 00:04:55,568 - Starting training for LSTM model... 2024-10-08 00:05:47,161 - Model training complete. MAE: 0.04162970831818715, MSE: 0.009068238198058645, R2: 0.9131405524668116 2024-10-08 00:05:47,220 - Model saved as lstm_model.pkl 2024-10-08 00:05:47,263 - LSTM Model - MAE: 0.04162970831818715, MSE: 0.009068238198058645, R2 Score: 0.9131405524668116 2024-10-08 00:12:48,546 - Building LSTM model with input shape (10, 207) 2024-10-08 00:12:48,548 - Starting training for LSTM model... 2024-10-08 00:13:32,477 - Model training complete. MAE: 0.038580160142659546, MSE: 0.008547108226241804, R2: 0.9181321572809295 PROJECT: Objective Setting This team project builds on the knowledge and experience you gained in the individual assignment involving Kafka for real-time traffic prediction. Now, as a team, you will work on a more comprehensive project using the METR-LA dataset, focusing on the operationalization of machine learning models in a real-world context. This assignment will involve experimentation, deployment, and monitoring phases, leveraging advanced tools such as Kubeflow, Kubernetes, and Evidently. It will also include a video presentation and a class Q&A session. Learning Outcomes By the end of this assignment, you will: Experiment with Machine Learning Models: Build and evaluate two models on the METR-LA dataset, focusing on LSTM and another selected model (ARIMA, RNN, GRU, or DCRNN). Deploy Models Using Kubernetes: Containerize your model and deploy it in a scalable, real-time environment using Docker and Kubernetes. Monitor Models in Production: Implement real-time monitoring using a pre-configured Evidently dashboard to ensure the model's performance remains robust. Communicate Insights Effectively: Present key findings and demonstrate your model through a video presentation and live Q&A session. Scenario Description and Context The Urban Mobility Challenge: As urban populations grow, traffic congestion becomes a major challenge. Your task is to develop, deploy, and monitor a predictive model for traffic flow using the METR-LA dataset. The model will be used to optimize traffic management systems in Los Angeles, improving the flow of vehicles and reducing congestion. Dataset Description: Name: METR-LA Traffic Dataset Source: GitHub - METR-LA Dataset Description: This dataset contains speed readings from 207 loop detectors across Los Angeles, recorded every five minutes. It provides a rich temporal sequence of traffic conditions, making it ideal for time-series forecasting and machine learning applications. Data links - https://www.kaggle.com/datasets/annnnguyen/metr-la-datasetLinks to an external site. {METR-LA.h5) OR https://github.com/tijsmaas/TrafficPrediction/blob/master/data/metr-la/metr-la.h5Links to an external site. Assignment Phases and Deliverables Phase 1: Model Experimentation Using Kubeflow (30 Points) Objective: Experiment with LSTM and one other selected model (ARIMA, RNN, GRU, or DCRNN) on the METR-LA dataset using Kubeflow. Tasks: Model Building: Implement LSTM and one other selected model using the provided starter notebook. Implement your models using TensorFlow, PyTorch, or any suitable framework. Experiment Tracking: Use Kubeflow to track your experiments, logging parameters, metrics, and artifacts for each model. Model Evaluation: Evaluate models using metrics like MAE, RMSE, and R². Visualize and compare model performance. Deliverables: Jupyter Notebook: Name the file model_experimentation.ipynb. The notebook should include code and documentation for model experimentation, including LSTM and one other selected model (ARIMA, RNN, GRU, or DCRNN). Clearly document the steps taken, including data preprocessing, model building, experimentation, and evaluation. Kubeflow Logs: Export logs from Kubeflow tracking the experimentation process. Save the logs as a PDF named kubeflow_logs.pdf. The logs should include parameters, metrics, and artifacts for each model. Model Comparison Report: Create a detailed report summarizing the performance and findings of the two models. Save the report as a PDF named model_comparison_report.pdf. The report should include visualizations comparing the models based on MAE, RMSE, R², and any other relevant metrics. Tools to Use: Python Libraries: TensorFlow, PyTorch, Scikit-learn Kubeflow: For managing and tracking ML experiments. Starter Notebook: Provided (see file starter_notebook_phase_1.py). Phase 2: Model Deployment Using Kubernetes (20 Points) Objective: Deploy the selected model using Docker and Kubernetes. Tasks: Model Packaging: Containerize the selected model using the provided Dockerfile template. Create RESTful API endpoints for serving predictions. Deployment and Orchestration: Deploy your Docker container using Kubernetes. Ensure scalability and reliability. Deliverables: Docker Image: Containerize your selected model using Docker. Push the Docker image to DockerHub. Provide a link to the DockerHub repository in a text file named docker_image_link.txt. Kubernetes Deployment Files: Include all YAML configuration files required for Kubernetes deployment. Ensure these files are well-documented and include comments explaining key configurations. Zip all the YAML files into a single file named kubernetes_deployment_files.zip. API Documentation: Write a detailed document explaining how to interact with the deployed model via the RESTful API.

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Include endpoint descriptions, sample requests, and expected responses. Save the document as a PDF named api_documentation.pdf. Starter Notebook: Provided (see file starter_notebook_phase_2.py). Phase 3: Model Monitoring Using Evidently (20 Points) Objective: Set up and implement model monitoring using a pre-configured Evidently dashboard. Tasks: Monitoring Setup: Integrate the provided Evidently monitoring dashboard with your deployed model. Track performance metrics and set up alerts for model drift. Video Presentation: Create a 5-minute video summarizing the key insights from your project. Demonstrate your model's performance and explain its real-world implications. Upload the video to the course platform. Deliverables: Docker Image: Containerize your selected model using Docker. Push the Docker image to DockerHub. Provide a link to the DockerHub repository in a text file named docker_image_link.txt. Kubernetes Deployment Files: Include all YAML configuration files required for Kubernetes deployment. Ensure these files are well-documented and include comments explaining key configurations. Zip all the YAML files into a single file named kubernetes_deployment_files.zip. API Documentation: Write a detailed document explaining how to interact with the deployed model via the RESTful API. Include endpoint descriptions, sample requests, and expected responses. Save the document as a PDF named api_documentation.pdf. Video Presentation: Upload the video to the course platform and provide a link in a text file named video_presentation_link.txt. Pre-configured Dashboard Code: Provided (see file evidently_monitoring_setup.py). Phase 4: Presentation (20 Points) Objective: Effectively communicate your project insights and demonstrate your model through a video and a live class presentation. Tasks: Class Presentation and Q&A: Present your project/play the video in a live class session. Be prepared to answer questions from fellow students and the instructor. Ensure that all team members participate in the presentation. Deliverables: Class Presentation and Q&A: Prepare a slide deck for your video/live class presentation. All team members should be ready to participate in the Q&A session. Submit the slide deck as a PDF named class_presentation.pdf. Submission Format: GitHub Repository: Create a public GitHub repository for the project. Include all code, documentation, and deliverables in the repository. Structure the repository with clear folder names corresponding to each phase. Submit the link to the GitHub repository in a text file named github_repository_link.txt. DockerHub Repository: Ensure the Docker image is accessible and correctly configured in DockerHub. Provide the link as detailed above. Evaluation Criteria Phase 1: Model Experimentation (25 Points): Quality of experimentation, use of Kubeflow, and thoroughness of the comparison report. Phase 2: Model Deployment (20 Points): Correct deployment using Docker and Kubernetes, and the clarity of API documentation. Phase 3: Model Monitoring (20 Points): Proper implementation of the monitoring dashboard and the robustness of the maintenance plan. Phase 4: Presentation (20 Points): Quality and effectiveness of the class presentation. Final Report (15 Points): Clarity, organization, and thoroughness of the final report and video PHASE 1 CODE: import numpy as np import pandas as pd from sklearn.preprocessing import MinMaxScaler from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score import mlflow import mlflow.keras from tensorflow.keras.models import Sequential from tensorflow.keras.layers import LSTM, GRU, Dense import logging import h5py # Step 1: Set up logging to a file (train.log) logging.basicConfig(filename='train.log', level=logging.INFO, format='%(asctime)s - %(message)s') # Step 2: Load and preprocess the dataset def load_data(): with h5py.File(r'C:\foai\94879-starter-code-Team-Project\metr-la.h5', 'r') as f: df_group = f['df'] # Load the column names (features) from 'block0_items' columns = list(df_group['block0_items'][:].astype(str)) # Load the actual data from 'block0_values' data = df_group['block0_values'][:] # Convert the data into a DataFrame df = pd.DataFrame(data, columns=columns) return df def preprocess_data(data): # Use MinMaxScaler to normalize the data scaler = MinMaxScaler() data_scaled = scaler.fit_transform(data) return data_scaled, scaler # Step 3: Create sequences for time-series forecasting def create_sequences(data, time_steps=10): X, y = [], [] for i in range(len(data) - time_steps): X.append(data[i:i + time_steps]) # Sequence of features y.append(data[i + time_steps, 0]) # Predict the next value for the first feature return np.array(X), np.array(y) # Step 4: Define the LSTM and GRU models def build_lstm_model(input_shape): model = Sequential() model.add(LSTM(64, input_shape=input_shape, return_sequences=True)) model.add(LSTM(32)) model.add(Dense(1)) model.compile(optimizer='adam', loss='mse', metrics=['mae']) return model def build_gru_model(input_shape): model = Sequential() model.add(GRU(64, input_shape=input_shape, return_sequences=True)) model.add(GRU(32)) model.add(Dense(1)) model.compile(optimizer='adam', loss='mse', metrics=['mae']) return model # Step 5: Train and evaluate the model, and log to MLflow def train_and_evaluate(model_type="LSTM"): # Load and preprocess the data data = load_data() data_scaled, scaler = preprocess_data(data) # Create sequences time_steps = 10 X, y = create_sequences(data_scaled, time_steps) # Check the number of features in the dataset num_features = X.shape[2] if len(X.shape) > 2 else 1 # Reshape the data to (samples, time_steps, num_features) X = X.reshape((X.shape[0], X.shape[1], num_features)) # Use actual number of features input_shape = (X.shape[1], X.shape[2]) # Start an MLflow run with mlflow.start_run(): # Select and build the model if model_type == "LSTM": model = build_lstm_model(input_shape) logging.info(f"Building LSTM model with input shape {input_shape}") elif model_type == "GRU": model = build_gru_model(input_shape) logging.info(f"Building GRU model with input shape {input_shape}") # Log the model type as a parameter mlflow.log_param("model_type", model_type) logging.info(f"Starting training for {model_type} model...") # Train the model history = model.fit(X, y, epochs=5, batch_size=32, validation_split=0.2) # Make predictions and calculate metrics y_pred = model.predict(X) mae = mean_absolute_error(y, y_pred) mse = mean_squared_error(y, y_pred) r2 = r2_score(y, y_pred) # Log metrics mlflow.log_metric("mae", mae) mlflow.log_metric("mse", mse) mlflow.log_metric("r2_score", r2) logging.info(f"Model training complete. MAE: {mae}, MSE: {mse}, R2: {r2}") # Log the trained model to MLflow mlflow.keras.log_model(model, "traffic_prediction_model") # Upload the train.log file as an artifact mlflow.log_artifact("train.log") logging.info(f"{model_type} Model - MAE: {mae}, MSE: {mse}, R2 Score: {r2}") if __name__ == "__main__": # Example of running the training for LSTM train_and_evaluate(model_type="LSTM") # You can also train the GRU model by passing model_type="GRU" # train_and_evaluate(model_type="GRU") PHASE 2 CODE: # Phase 2: Model Deployment using Docker and Kubernetes # This starter notebook will guide you through deploying your trained model as a RESTful API using Flask. # Import necessary libraries from flask import Flask, request, jsonify import numpy as np import tensorflow as tf # Load the trained model (ensure your model is saved from Phase 1) def load_model(model_name='lstm_model.h5'): return tf.keras.models.load_model(model_name) # Initialize Flask app app = Flask(__name__) model = load_model() # Replace 'lstm_model.h5' with the appropriate model file @app.route('/predict', methods=['POST']) def predict(): # Parse input data data = request.get_json(force=True) prediction_input = np.array(data['input']).reshape(1, -1) # Adjust input shape as per your model prediction = model.predict(prediction_input).tolist() return jsonify({'prediction': prediction}) if __name__ == '__main__': # Run the Flask app app.run(host='0.0.0.0', port=5000) # TODO: Build a Dockerfile for this Flask app and deploy using Kubernetes

Thought for 10 seconds

Certainly! Let's help you finish Phase 2 by completing your model deployment using Docker and Kubernetes. We'll go through the following steps:

1. **Finalize the Flask App**: Ensure the Flask app correctly loads your trained model and handles input data appropriately.
2. **Create a Dockerfile**: Write a Dockerfile to containerize your Flask app.
3. **Build and Push Docker Image**: Build the Docker image and push it to DockerHub.
4. **Write Kubernetes Deployment Files**: Create the necessary YAML files for deploying your app on Kubernetes.
5. **API Documentation**: Provide documentation on how to interact with your API.

Let's dive into each step.

---

1. Finalize the Flask App

a. Load the Trained Model

First, ensure that your Flask app loads the trained model correctly. Since you saved your model using `mlflow.keras.log_model`, you'll need to adjust how you load it.

Updated `load_model` Function:

`python`

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Copy code

```
import mlflow.keras

def load_model(model_uri='models:/traffic_prediction_model/Production'):
    return mlflow.keras.load_model(model_uri)
```

This assumes you've registered your model in the MLflow Model Registry and set it to the 'Production' stage.

If you saved your model locally, adjust the `model_uri` accordingly:

python

Copy code

```
def load_model(model_path='models/traffic_prediction_model'):
    return mlflow.keras.load_model(model_path)
```

b. Update the Predict Endpoint

Adjust the `predict` function to handle input data as per your model's requirements.

Complete Flask App (`app.py`):

python

Copy code

```
from flask import Flask, request, jsonify
import numpy as np
import mlflow.keras
import tensorflow as tf

# Load the trained model
def load_model(model_uri='traffic_prediction_model'):
    return mlflow.keras.load_model(model_uri)

# Initialize Flask app
app = Flask(__name__)
model = load_model()  # Adjust model_uri if necessary

@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Parse input data
        data = request.get_json(force=True)
        input_data = data['input']  # Expecting a list of lists

        # Convert input data to numpy array and reshape
        prediction_input = np.array(input_data)
        # Reshape input to match the model's expected input
        # For example, (samples, time_steps, features)
        prediction_input = prediction_input.reshape((1, 10, -1))  # Adjust time_steps and features as per your model

        # Make prediction
        prediction = model.predict(prediction_input)

        # Return the prediction as a list
        return jsonify({'prediction': prediction.tolist()})

    except Exception as e:
        return jsonify({'error': str(e)})

if __name__ == '__main__':
    # Run the Flask app
    app.run(host='0.0.0.0', port=5000)
```

Notes:

- Ensure that the input data sent to the `/predict` endpoint matches the expected shape `(samples, time_steps, features)` of your model.
- Adjust the `reshape` parameters based on your model's input shape.

c. Test the Flask App Locally

Before containerizing, test your Flask app locally to ensure it works as expected.

Testing the App:

1. Run the Flask app:

   bash

   Copy code

   ```
   python app.py
   ```

2. Send a test POST request using `curl` or a tool like Postman:

   bash

   Copy code

   ```
   curl -X POST -H "Content-Type: application/json" -d '{"input": [[0.1, 0.2, ..., 0.207]]}' http://localhost:5000/predict
   ```

   Replace `...` with actual values to match your model's expected input.

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

2. Create a Dockerfile

Create a `Dockerfile` to containerize your Flask app.

Dockerfile:

Dockerfile

Copy code

```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file into the container
COPY requirements.txt .

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the app's code into the container
COPY . .

# Expose port 5000 for the Flask app
EXPOSE 5000

# Define environment variable
ENV FLASK_APP=app.py

# Command to run the Flask app
CMD ["flask", "run", "--host=0.0.0.0", "--port=5000"]
```

Notes:

- Ensure you have a `requirements.txt` file with all necessary dependencies.
- Copy your model files into the container if they are saved locally.
- If your model is logged in MLflow and stored remotely, ensure network access in your container to load it.

Example `requirements.txt`:

makefile

Copy code

```
Flask==2.0.1
mlflow==2.16.2
tensorflow==2.17.0
numpy==1.26.4
```

---

3. Build and Push Docker Image

a. Build the Docker Image

Run the following command in the directory containing your `Dockerfile`:

bash

Copy code

```
docker build -t your-dockerhub-username/traffic-prediction-app:latest .
```

b. Test the Docker Image Locally

Run the Docker container:

bash

Copy code

```
docker run -p 5000:5000 your-dockerhub-username/traffic-prediction-app:latest
```

Test the API as before to ensure it works inside the container.

c. Push the Docker Image to DockerHub

1. Log in to DockerHub:

   bash

   Copy code

   ```
   docker login
   ```

2. Push the image:

   bash

   Copy code

   ```
   docker push your-dockerhub-username/traffic-prediction-app:latest
   ```

3. Provide the DockerHub repository link in `docker_image_link.txt`.

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

4. Write Kubernetes Deployment Files

Create the necessary YAML files for deploying your app on Kubernetes.

a. Deployment Configuration (`deployment.yaml`)

yaml

Copy code

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: traffic-prediction-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: traffic-prediction
  template:
    metadata:
      labels:
        app: traffic-prediction
    spec:
      containers:
      - name: traffic-prediction-container
        image: your-dockerhub-username/traffic-prediction-app:latest
        ports:
        - containerPort: 5000
```

b. Service Configuration (`service.yaml`)

yaml

Copy code

```
apiVersion: v1
kind: Service
metadata:
  name: traffic-prediction-service
spec:
  selector:
    app: traffic-prediction
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5000
  type: LoadBalancer
```

Notes:

- The `Deployment` defines how many replicas (pods) you want.
- The `Service` exposes your app to external traffic.

c. Apply the Kubernetes Configurations

Deploy your application to the Kubernetes cluster:

bash

Copy code

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

d. Verify the Deployment

Check the status of your pods and services:

bash

Copy code

```
kubectl get pods
kubectl get services
```

Zip all YAML files into `kubernetes_deployment_files.zip`.

---

5. API Documentation

Provide clear documentation on how to interact with your API.

API Documentation (`api_documentation.pdf`):

Traffic Prediction API Documentation

Endpoint: `/predict`

- **URL:** `/predict`
- **Method:** `POST`
- **Content-Type:** `application/json`

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

- **Description:** Predicts traffic speed based on input data.

Request Body

- **Parameter:** `input`
- **Type:** List of lists (2D array)
- **Description:** Time series data for prediction. Should be shaped as `(time_steps, features)`.

Example:

json

Copy code

```
{
  "input": [
    [0.1, 0.2, ..., 0.207],
    [0.2, 0.3, ..., 0.208],
    ...
    [0.10, 0.11, ..., 0.216]
  ]
}
```

- The above example assumes `time_steps=10` and `features=207`.

Response

- **Success Response:**
  - **Code:** `200 OK`
  - **Content:**

    json

    Copy code

    ```
    {
      "prediction": [[0.5, 0.6, ..., 0.7]]
    }
    ```

- **Error Response:**
  - **Code:** `400 Bad Request`
  - **Content:**

    json

    Copy code

    ```
    {
      "error": "Error message detailing what went wrong."
    }
    ```

Sample Request Using `curl`

bash

Copy code

```
curl -X POST -H "Content-Type: application/json" \
-d '{"input": [[0.1, 0.2, ..., 0.207], ..., [0.10, 0.11, ..., 0.216]]}' \
http://your-service-url/predict
```

Note: Replace `your-service-url` with the actual URL of your service.

Additional Tips

- **Model Loading:** If your model is saved locally (e.g., as `model.h5`), make sure it's included in your Docker image by copying it in the `Dockerfile`.
- **Environment Variables:** If you need to pass any environment variables (e.g., for MLflow tracking URIs), define them in your `Deployment` YAML under `env`.
- **Logging:** Consider adding logging to your Flask app to help with debugging.

Conclusion

By following these steps, you should be able to:

- Finalize your Flask app to serve predictions.
- Containerize your app using Docker.
- Deploy your containerized app on Kubernetes.
- Provide clear API documentation for users.

Make sure to test each step thoroughly to ensure everything works as expected before moving on to the next. If you encounter any issues, check the logs from your Flask app, Docker container, and Kubernetes pods to diagnose and fix problems.

Remember to:

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

- Update the placeholders like `your-dockerhub-username` and `your-service-url` with your actual information.
- Ensure all files are correctly named and included in your GitHub repository.
- Document any assumptions or deviations from the starter code in your submission.

Feel free to ask if you need further clarification or assistance with any of the steps!

You said:

I believe my model was saved here. Once I ran Phase 1 CODE. Is Phase code 1 right? I assume so but very important so I can get Phase 2 complete. Please give me the entire code needed to get Phase 2 complete. Also give me other code needed like docker file, etc. Please take your time to think about everything I need to do to finish Phase 2 code. I currently have http://127.0.0.1:5000/ running in URL that shows me the saved model from Phase 1 CODE. file:///C:/foai/mlruns/0/c58955aa3bbf43a9b838abc4769280b2/artifacts/traffic_prediction_model/data/model.keras Phase 1 CODE: import numpy as np import pandas as pd from sklearn.preprocessing import MinMaxScaler from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score import mlflow import mlflow.keras from tensorflow.keras.models import Sequential from tensorflow.keras.layers import LSTM, GRU, Dense import logging import h5py # Step 1: Set up logging to a file (train.log) logging.basicConfig(filename='train.log', level=logging.INFO, format='%(asctime)s - %(message)s') # Step 2: Load and preprocess the dataset def load_data(): with h5py.File(r'C:\foai\94879-starter-code-Team-Project\metr-la.h5', 'r') as f: df_group = f['df'] # Load the column names (features) from 'block0_items' columns = list(df_group['block0_items'][:].astype(str)) # Load the actual data from 'block0_values' data = df_group['block0_values'][:] # Convert the data into a DataFrame df = pd.DataFrame(data, columns=columns) return df def preprocess_data(data): # Use MinMaxScaler to normalize the data scaler = MinMaxScaler() data_scaled = scaler.fit_transform(data) return data_scaled, scaler # Step 3: Create sequences for time-series forecasting def create_sequences(data, time_steps=10): X, y = [], [] for i in range(len(data) - time_steps): X.append(data[i:i + time_steps]) # Sequence of features y.append(data[i + time_steps, 0]) # Predict the next value for the first feature return np.array(X), np.array(y) # Step 4: Define the LSTM and GRU models def build_lstm_model(input_shape): model = Sequential() model.add(LSTM(64, input_shape=input_shape, return_sequences=True)) model.add(LSTM(32)) model.add(Dense(1)) model.compile(optimizer='adam', loss='mse', metrics=['mae']) return model def build_gru_model(input_shape): model = Sequential() model.add(GRU(64, input_shape=input_shape, return_sequences=True)) model.add(GRU(32)) model.add(Dense(1)) model.compile(optimizer='adam', loss='mse', metrics=['mae']) return model # Step 5: Train and evaluate the model, and log to MLflow def train_and_evaluate(model_type="LSTM"): # Load and preprocess the data data = load_data() data_scaled, scaler = preprocess_data(data) # Create sequences time_steps = 10 X, y = create_sequences(data_scaled, time_steps) # Check the number of features in the dataset num_features = X.shape[2] if len(X.shape) > 2 else 1 # Reshape the data to (samples, time_steps, num_features) X = X.reshape((X.shape[0], X.shape[1], num_features)) # Use actual number of features input_shape = (X.shape[1], X.shape[2]) # Start an MLflow run with mlflow.start_run(): # Select and build the model if model_type == "LSTM": model = build_lstm_model(input_shape) logging.info(f"Building LSTM model with input shape {input_shape}") elif model_type == "GRU": model = build_gru_model(input_shape) logging.info(f"Building GRU model with input shape {input_shape}") # Log the model type as a parameter mlflow.log_param("model_type", model_type) logging.info(f"Starting training for {model_type} model...") # Train the model history = model.fit(X, y, epochs=5, batch_size=32, validation_split=0.2) # Make predictions and calculate metrics y_pred = model.predict(X) mae = mean_absolute_error(y, y_pred) mse = mean_squared_error(y, y_pred) r2 = r2_score(y, y_pred) # Log metrics mlflow.log_metric("mae", mae) mlflow.log_metric("mse", mse) mlflow.log_metric("r2_score", r2) logging.info(f"Model training complete. MAE: {mae}, MSE: {mse}, R2: {r2}") # Log the trained model to MLflow mlflow.keras.log_model(model, "traffic_prediction_model") # Upload the train.log file as an artifact mlflow.log_artifact("train.log") logging.info(f"{model_type} Model - MAE: {mae}, MSE: {mse}, R2 Score: {r2}") if __name__ == "__main__": # Example of running the training for LSTM train_and_evaluate(model_type="LSTM") # You can also train the GRU model by passing model_type="GRU" # train_and_evaluate(model_type="GRU") PHASE 2 CODE: # Phase 2: Model Deployment using Docker and Kubernetes # This starter notebook will guide you through deploying your trained model as a RESTful API using Flask. # Import necessary libraries from flask import Flask, request, jsonify import numpy as np import tensorflow as tf import logging import h5py import joblib # Load the trained model (ensure your model is saved from Phase 1) def load_model(model_name='lstm_model.pkl'): try: return joblib.load(model_name) except Exception as e: print(f"Error loading model: {e}") return None # Initialize Flask app app = Flask(__name__) model = load_model() # Replace 'lstm_model.h5' with the appropriate model file @app.route('/predict', methods=['POST']) def predict(): # Parse input data data = request.get_json(force=True) prediction_input = np.array(data['input']).reshape(1, -1) # Adjust input shape as per your model prediction = model.predict(prediction_input).tolist() return jsonify({'prediction': prediction}) if __name__ == '__main__': # Run the Flask app app.run(host='0.0.0.0', port=5000) # TODO: Build a Dockerfile for this Flask app and deploy using Kubernetes CURRENT DOCKER FILE. WILL NEED ADJUSTED COD: # Use an official Python runtime as a parent image FROM python:3.8-slim # Set the working directory inside the container WORKDIR /app # Copy the requirements file into the container at /app COPY requirements.txt /app/ # Install any required packages specified in requirements.txt RUN pip install --no-cache-dir -r requirements.txt # Copy the rest of the application code into the container COPY . /app # Expose the port that the Flask app will run on EXPOSE 5000 # Command to run the application CMD ["python", "starter_notebook_phase_2.py"]

Thought for 48 seconds

Certainly! Let's help you finish Phase 2 by ensuring your model from Phase 1 is correctly saved and loaded, adjusting your Flask app, and providing all necessary code, including the Dockerfile, to complete Phase 2.

Phase 1: Adjustments

Your Phase 1 code is mostly correct, but to facilitate loading the model in Phase 2, we need to save the trained model in a format that's easy to load in the Flask app. Since you're using Keras, saving the model as an `.h5` file is appropriate.

Modified Phase 1 Code

Add the following line to save your trained model after logging it to MLflow:

python

Copy code

```python
# Save the model locally as 'lstm_model.h5'
model.save('lstm_model.h5')
```

Complete Adjusted Phase 1 Code:

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

python

Copy code

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import mlflow
import mlflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, GRU, Dense
import logging
import h5py

# Step 1: Set up logging to a file (train.log)
logging.basicConfig(filename='train.log', level=logging.INFO, format='%(asctime)s - %(message)s')

# Step 2: Load and preprocess the dataset
def load_data():
    with h5py.File(r'C:\foai\94879-starter-code-Team-Project\metr-la.h5', 'r') as f:
        df_group = f['df']

        # Load the column names (features) from 'block0_items'
        columns = list(df_group['block0_items'][:].astype(str))

        # Load the actual data from 'block0_values'
        data = df_group['block0_values'][:]

        # Convert the data into a DataFrame
        df = pd.DataFrame(data, columns=columns)

        return df

def preprocess_data(data):
    # Use MinMaxScaler to normalize the data
    scaler = MinMaxScaler()
    data_scaled = scaler.fit_transform(data)
    return data_scaled, scaler

# Step 3: Create sequences for time-series forecasting
def create_sequences(data, time_steps=10):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:i + time_steps])  # Sequence of features
        y.append(data[i + time_steps, 0])  # Predict the next value for the first feature
    return np.array(X), np.array(y)

# Step 4: Define the LSTM and GRU models
def build_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(64, input_shape=input_shape, return_sequences=True))
    model.add(LSTM(32))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

def build_gru_model(input_shape):
    model = Sequential()
    model.add(GRU(64, input_shape=input_shape, return_sequences=True))
    model.add(GRU(32))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse', metrics=['mae'])
    return model

# Step 5: Train and evaluate the model, and log to MLflow
def train_and_evaluate(model_type="LSTM"):
    # Load and preprocess the data
    data = load_data()
    data_scaled, scaler = preprocess_data(data)

    # Create sequences
    time_steps = 10
    X, y = create_sequences(data_scaled, time_steps)

    # Check the number of features in the dataset
    num_features = X.shape[2] if len(X.shape) > 2 else 1

    # Reshape the data to (samples, time_steps, num_features)
    X = X.reshape((X.shape[0], X.shape[1], num_features))  # Use actual number of features

    input_shape = (X.shape[1], X.shape[2])

    # Start an MLflow run
    with mlflow.start_run():
        # Select and build the model
        if model_type == "LSTM":
            model = build_lstm_model(input_shape)
            logging.info(f"Building LSTM model with input shape {input_shape}")
        elif model_type == "GRU":
            model = build_gru_model(input_shape)
            logging.info(f"Building GRU model with input shape {input_shape}")

        # Log the model type as a parameter
        mlflow.log_param("model_type", model_type)
        logging.info(f"Starting training for {model_type} model...")

        # Train the model
        history = model.fit(X, y, epochs=5, batch_size=32, validation_split=0.2)

        # Make predictions and calculate metrics
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
        y_pred = model.predict(X)
        mae = mean_absolute_error(y, y_pred)
        mse = mean_squared_error(y, y_pred)
        r2 = r2_score(y, y_pred)

        # Log metrics
        mlflow.log_metric("mae", mae)
        mlflow.log_metric("mse", mse)
        mlflow.log_metric("r2_score", r2)

        logging.info(f"Model training complete. MAE: {mae}, MSE: {mse}, R2: {r2}")

        # Log the trained model to MLflow
        mlflow.keras.log_model(model, "traffic_prediction_model")

        # Save the model locally as 'lstm_model.h5'
        model.save('lstm_model.h5')

        # Upload the train.log file as an artifact
        mlflow.log_artifact("train.log")

        logging.info(f"{model_type} Model - MAE: {mae}, MSE: {mse}, R2 Score: {r2}")

if __name__ == "__main__":
    # Example of running the training for LSTM
    train_and_evaluate(model_type="LSTM")

    # You can also train the GRU model by passing model_type="GRU"
    # train_and_evaluate(model_type="GRU")
```

Phase 2: Model Deployment

1. Adjusting the Flask App

Now that your model is saved as `lstm_model.h5`, you can load it in your Flask app.

Complete Flask App (`app.py`):

python

Copy code

```python
from flask import Flask, request, jsonify
import numpy as np
from tensorflow.keras.models import load_model

# Load the trained model
def load_trained_model(model_path='lstm_model.h5'):
    return load_model(model_path)

# Initialize Flask app
app = Flask(__name__)
model = load_trained_model()

@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Parse input data
        data = request.get_json(force=True)
        input_data = data['input']  # Expecting a 2D list (time_steps x num_features)

        # Convert input data to numpy array
        prediction_input = np.array(input_data)

        # Reshape input to match the model's expected input shape
        # The model expects input shape: (samples, time_steps, num_features)
        prediction_input = prediction_input.reshape((1, prediction_input.shape[0], prediction_input.shape[1]))

        # Make prediction
        prediction = model.predict(prediction_input)

        # Return the prediction as a list
        return jsonify({'prediction': prediction.tolist()})

    except Exception as e:
        return jsonify({'error': str(e)})

if __name__ == '__main__':
    # Run the Flask app
    app.run(host='0.0.0.0', port=5000)
```

Notes:

- The `load_trained_model` function loads `lstm_model.h5`.
- The `/predict` endpoint accepts JSON data with an `input` key containing a 2D list.
- The input data is reshaped to match the model's expected input shape.

2. Testing the Flask App Locally

Before containerizing, test your Flask app:

1. Run the Flask app:

   bash

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Copy code

```
python app.py
```

2. Send a test POST request using `curl` or Postman:

   bash

   Copy code

   ```
   curl -X POST -H "Content-Type: application/json" -d '{"input": [[0.1, 0.2, ..., 0.207], ..., [0.1, 0.2, ..., 0.207]]}' http://127.0
   ```

   Replace `...` with actual numerical values, ensuring you have a total of 10 time steps (since `time_steps=10`) and 207 features per time step.

---

3. Creating the Dockerfile

Dockerfile:

dockerfile

Copy code

```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the requirements file into the container
COPY requirements.txt /app/

# Install any required packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code into the container
COPY . /app

# Expose the port that the Flask app will run on
EXPOSE 5000

# Command to run the application
CMD ["python", "app.py"]
```

requirements.txt:

makefile

Copy code

```
Flask==2.0.1
tensorflow==2.17.0
numpy==1.26.4
```

Notes:

- Ensure `lstm_model.h5`, `app.py`, and `requirements.txt` are in the same directory.
- The Dockerfile sets up the environment, installs dependencies, and runs your Flask app.

---

4. Building and Pushing the Docker Image

a. Build the Docker Image

In the directory containing your `Dockerfile`, run:

bash

Copy code

```
docker build -t your-dockerhub-username/traffic-prediction-app:latest .
```

b. Test the Docker Image Locally

Run the Docker container:

bash

Copy code

```
docker run -p 5000:5000 your-dockerhub-username/traffic-prediction-app:latest
```

Test the API as before to ensure it works inside the container.

c. Push the Docker Image to DockerHub

1. Log in to DockerHub:

   bash

   Copy code

   ```
   docker login
   ```

2. Push the image:

   bash

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Copy code

```
docker push your-dockerhub-username/traffic-prediction-app:latest
```

3. Provide the DockerHub repository link in a text file named `docker_image_link.txt`.

---

5. Kubernetes Deployment Files

a. Deployment Configuration (`deployment.yaml`):

yaml

Copy code

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: traffic-prediction-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: traffic-prediction
  template:
    metadata:
      labels:
        app: traffic-prediction
    spec:
      containers:
      - name: traffic-prediction-container
        image: your-dockerhub-username/traffic-prediction-app:latest
        ports:
        - containerPort: 5000
```

b. Service Configuration (`service.yaml`):

yaml

Copy code

```
apiVersion: v1
kind: Service
metadata:
  name: traffic-prediction-service
spec:
  selector:
    app: traffic-prediction
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5000
  type: LoadBalancer
```

Notes:

- Replace `your-dockerhub-username` with your actual DockerHub username.
- The `Deployment` manages your app instances (pods), and the `Service` exposes your app.

c. Apply the Kubernetes Configurations

Deploy your application:

bash

Copy code

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

d. Verify the Deployment

Check the status:

bash

Copy code

```
kubectl get pods
kubectl get services
```

e. Package the YAML Files

Zip the YAML files into `kubernetes_deployment_files.zip`.

---

6. API Documentation

Create a document named `api_documentation.pdf` with the following content:

Traffic Prediction API Documentation

Endpoint: `/predict`

- **URL:** `/predict`

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

- **Method:** `POST`
- **Content-Type:** `application/json`
- **Description:** Predicts traffic speed based on input data.

Request Body

- **Parameter:** `input`
- **Type:** List of lists (2D array)
- **Description:** Time series data for prediction. Should be shaped as `(time_steps, num_features)`.

Example Request:

json
Copy code

```
{
  "input": [
    [0.1, 0.2, ..., 0.207],
    [0.2, 0.3, ..., 0.208],
    ... (10 time steps in total)
  ]
}
```

- The example assumes `time_steps=10` and `num_features=207`.

Response

- **Success Response:**
  - **Code:** `200 OK`
  - **Content:**

    json
    Copy code

    ```
    {
      "prediction": [[0.5]]
    }
    ```

- **Error Response:**
  - **Code:** `400 Bad Request`
  - **Content:**

    json
    Copy code

    ```
    {
      "error": "Error message detailing what went wrong."
    }
    ```

Sample Request Using `curl`:

bash
Copy code

```
curl -X POST -H "Content-Type: application/json" \
-d '{"input": [[0.1, 0.2, ..., 0.207], ..., [0.1, 0.2, ..., 0.207]]}' \
http://your-service-url/predict
```

Note: Replace `your-service-url` with the actual URL of your service.

---

7. Additional Notes

- **Model Input Shape:** Ensure that the input data sent to the `/predict` endpoint is a 2D list with the shape `(time_steps, num_features)`.
- **Testing:** Always test your Flask app locally and inside the Docker container before deploying to Kubernetes.
- **Docker Image Accessibility:** Make sure your Docker image is publicly accessible or that your Kubernetes cluster has access to your Docker registry.

Summary of Deliverables

1. **Flask App (`app.py`):** Complete code for your Flask application.
2. **Dockerfile:** Adjusted Dockerfile to build your Docker image.
3. **Requirements File (`requirements.txt`):** List of dependencies for your Flask app.
4. **Docker Image Link (`docker_image_link.txt`):** Text file containing the link to your DockerHub repository.
5. **Kubernetes Deployment Files (`kubernetes_deployment_files.zip`):** Zipped YAML files for deploying your app.
6. **API Documentation (`api_documentation.pdf`):** Detailed documentation on how to interact with your API.

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Next Steps

1. **Ensure All Files Are in Your Project Directory:**
    - `app.py`
    - `Dockerfile`
    - `requirements.txt`
    - `lstm_model.h5`
    - Kubernetes YAML files
2. **Test Your Application Thoroughly:**
    - Run `app.py` locally.
    - Build and run the Docker container locally.
    - Test the API endpoints.
3. **Push Your Docker Image to DockerHub:**
    - Ensure the image is accessible for deployment.
4. **Deploy to Kubernetes:**
    - Apply your deployment and service YAML files.
    - Verify the deployment and service status.
5. **Update Your GitHub Repository:**
    - Include all code and documentation.
    - Organize files according to project phases.
6. **Prepare for Phase 3 and Beyond:**
    - Start thinking about monitoring with Evidently.
    - Plan your video presentation and documentation.

Conclusion

By following these steps and using the provided code, you should be able to complete Phase 2 of your project successfully. Remember to adjust any placeholders with your actual information, test thoroughly at each step, and ensure all deliverables meet the assignment requirements.

If you have any further questions or need additional assistance, feel free to ask!

You said:

I think my app.py is what Phase 2 CODE is starter_notebook_phase_2.py I got this error when running starter_notebook_phase_2.py CODE: from flask import Flask, request, jsonify import numpy as np from tensorflow.keras.models import load_model # Load the trained model def load_trained_model(model_path='lstm_model.h5'): return load_model(model_path) # Initialize Flask app app = Flask(__name__) model = load_trained_model() @app.route('/predict', methods=['POST']) def predict(): try: # Parse input data data = request.get_json(force=True) input_data = data['input'] # Expecting a 2D list (time_steps x num_features) # Convert input data to numpy array prediction_input = np.array(input_data) # Reshape input to match the model's expected input shape # The model expects input shape: (samples, time_steps, num_features) prediction_input = prediction_input.reshape((1, prediction_input.shape[0], prediction_input.shape[1])) # Make prediction prediction = model.predict(prediction_input) # Return the prediction as a list return jsonify({'prediction': prediction.tolist()}) except Exception as e: return jsonify({'error': str(e)}) if __name__ == '__main__': # Run the Flask app app.run(host='0.0.0.0', port=5000) OUTPUT: PS C:\foai> & C:/Users/16318/anaconda3/envs/opAI_VM/python.exe c:/foai/94879-starter-code-Team-Project/starter_notebook_phase_2.py 2024-10-08 00:30:30.860319: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`. 2024-10-08 00:30:31.622300: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`. 2024-10-08 00:30:34.263929: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations. To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags. Traceback (most recent call last): File "c:\foai\94879-starter-code-Team-Project\starter_notebook_phase_2.py", line 11, in <module> model = load_trained_model() ^^^^^^^^^^^^^^^^^^^^ File "c:\foai\94879-starter-code-Team-Project\starter_notebook_phase_2.py", line 7, in load_trained_model return load_model(model_path) ^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\16318\anaconda3\envs\opAI_VM\Lib\site-packages\keras\src\saving\saving_api.py", line 194, in load_model return legacy_h5_format.load_model_from_hdf5(^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\16318\anaconda3\envs\opAI_VM\Lib\site-packages\keras\src\legacy\saving\legacy_h5_format.py", line 155, in load_model_from_hdf5 **saving_utils.compile_args_from_training_config(^^ File "C:\Users\16318\anaconda3\envs\opAI_VM\Lib\site-packages\keras\src\legacy\saving\saving_utils.py", line 143, in compile_args_from_training_config loss = _deserialize_nested_config(losses.deserialize, loss_config) ^^^ File "C:\Users\16318\anaconda3\envs\opAI_VM\Lib\site-packages\keras\src\legacy\saving\saving_utils.py", line 202, in _deserialize_nested_config return deserialize_fn(config) ^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\16318\anaconda3\envs\opAI_VM\Lib\site-packages\keras\src\losses\__init__.py", line 149, in deserialize return serialization_lib.deserialize_keras_object(^^^ File "C:\Users\16318\anaconda3\envs\opAI_VM\Lib\site-packages\keras\src\saving\serialization_lib.py", line 575, in deserialize_keras_object return deserialize_keras_object(^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\16318\anaconda3\envs\opAI_VM\Lib\site-packages\keras\src\saving\serialization_lib.py", line 678, in deserialize_keras_object return _retrieve_class_or_fn(^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\16318\anaconda3\envs\opAI_VM\Lib\site-packages\keras\src\saving\serialization_lib.py", line 812, in _retrieve_class_or_fn raise TypeError(TypeError: Could not locate function 'mse'. Make sure custom classes are decorated with `@keras.saving.register_keras_serializable()`. Full object config: {'module': 'keras.metrics', 'class_name': 'function', 'config': 'mse', 'registered_name': 'mse'}

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Thought for 20 seconds

Certainly! The error you're encountering is due to how the loss function and metrics are specified when compiling your model in Phase 1. Let's address this issue step by step and provide all the necessary code to complete Phase 2.

Understanding the Error

The error message you're seeing:

sql

Copy code

```
TypeError: Could not locate function 'mse'. Make sure custom classes are decorated with `@keras.saving.register_keras_serializable()`. Fu
```

This error occurs because when you save a Keras model that was compiled with loss and metrics specified as strings (e.g., `'mse'`, `'mae'`), Keras sometimes cannot properly serialize and deserialize these strings when loading the model.

Solution Overview

To fix this issue, we'll:

1. **Modify Phase 1 Code**: Adjust the way the loss function and metrics are specified when compiling the model.
2. **Retrain and Save the Model**: Run the adjusted Phase 1 code to save a model that can be loaded without errors.
3. **Adjust Phase 2 Code**: Ensure your Flask app correctly loads the adjusted model.
4. **Provide All Necessary Code**: Include the complete code for Phase 2, including the Dockerfile and other required files.

Step 1: Modify Phase 1 Code

a. Update Model Compilation

In your Phase 1 code, when compiling the model, replace the string identifiers for loss and metrics with the actual functions from TensorFlow Keras.

Original Code:

python

Copy code

```python
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

Updated Code:

python

Copy code

```python
import tensorflow as tf  # Add this import at the top

model.compile(optimizer='adam',
              loss=tf.keras.losses.MeanSquaredError(),
              metrics=[tf.keras.metrics.MeanAbsoluteError()])
```

Explanation:

- Using the actual loss and metric functions ensures they are properly serialized when saving the model.
- This change prevents the deserialization error when loading the model in your Flask app.

b. Full Adjusted Phase 1 Code

Here's the complete adjusted Phase 1 code:

python

Copy code

```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import mlflow
import mlflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, GRU, Dense
import logging
import h5py
import tensorflow as tf  # Added import

# Step 1: Set up logging to a file (train.log)
logging.basicConfig(filename='train.log', level=logging.INFO, format='%(asctime)s - %(message)s')

# Step 2: Load and preprocess the dataset
def load_data():
    with h5py.File(r'C:\foai\94879-starter-code-Team-Project\metr-la.h5', 'r') as f:
        df_group = f['df']
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
        # Load the column names (features) from 'block0_items'
        columns = list(df_group['block0_items'][:].astype(str))

        # Load the actual data from 'block0_values'
        data = df_group['block0_values'][:]

        # Convert the data into a DataFrame
        df = pd.DataFrame(data, columns=columns)

        return df

def preprocess_data(data):
    # Use MinMaxScaler to normalize the data
    scaler = MinMaxScaler()
    data_scaled = scaler.fit_transform(data)
    return data_scaled, scaler

# Step 3: Create sequences for time-series forecasting
def create_sequences(data, time_steps=10):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:i + time_steps])  # Sequence of features
        y.append(data[i + time_steps, 0])  # Predict the next value for the first feature
    return np.array(X), np.array(y)

# Step 4: Define the LSTM and GRU models
def build_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(64, input_shape=input_shape, return_sequences=True))
    model.add(LSTM(32))
    model.add(Dense(1))
    model.compile(optimizer='adam',
                  loss=tf.keras.losses.MeanSquaredError(),
                  metrics=[tf.keras.metrics.MeanAbsoluteError()])
    return model

def build_gru_model(input_shape):
    model = Sequential()
    model.add(GRU(64, input_shape=input_shape, return_sequences=True))
    model.add(GRU(32))
    model.add(Dense(1))
    model.compile(optimizer='adam',
                  loss=tf.keras.losses.MeanSquaredError(),
                  metrics=[tf.keras.metrics.MeanAbsoluteError()])
    return model

# Step 5: Train and evaluate the model, and log to MLflow
def train_and_evaluate(model_type="LSTM"):
    # Load and preprocess the data
    data = load_data()
    data_scaled, scaler = preprocess_data(data)

    # Create sequences
    time_steps = 10
    X, y = create_sequences(data_scaled, time_steps)

    # Check the number of features in the dataset
    num_features = X.shape[2] if len(X.shape) > 2 else 1

    # Reshape the data to (samples, time_steps, num_features)
    X = X.reshape((X.shape[0], X.shape[1], num_features))  # Use actual number of features

    input_shape = (X.shape[1], X.shape[2])

    # Start an MLflow run
    with mlflow.start_run():
        # Select and build the model
        if model_type == "LSTM":
            model = build_lstm_model(input_shape)
            logging.info(f"Building LSTM model with input shape {input_shape}")
        elif model_type == "GRU":
            model = build_gru_model(input_shape)
            logging.info(f"Building GRU model with input shape {input_shape}")

        # Log the model type as a parameter
        mlflow.log_param("model_type", model_type)
        logging.info(f"Starting training for {model_type} model...")

        # Train the model
        history = model.fit(X, y, epochs=5, batch_size=32, validation_split=0.2)

        # Make predictions and calculate metrics
        y_pred = model.predict(X)
        mae = mean_absolute_error(y, y_pred)
        mse = mean_squared_error(y, y_pred)
        r2 = r2_score(y, y_pred)

        # Log metrics
        mlflow.log_metric("mae", mae)
        mlflow.log_metric("mse", mse)
        mlflow.log_metric("r2_score", r2)

        logging.info(f"Model training complete. MAE: {mae}, MSE: {mse}, R2: {r2}")

        # Log the trained model to MLflow
        mlflow.keras.log_model(model, "traffic_prediction_model")

        # Save the model locally as 'lstm_model.h5'
        model.save('lstm_model.h5')
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
        # Upload the train.log file as an artifact
        mlflow.log_artifact("train.log")

        logging.info(f"{model_type} Model - MAE: {mae}, MSE: {mse}, R2 Score: {r2}")

if __name__ == "__main__":
    # Example of running the training for LSTM
    train_and_evaluate(model_type="LSTM")
```

Step 2: Retrain and Save the Model

- **Run the adjusted Phase 1 code** to retrain your model.
- Ensure that `lstm_model.h5` is saved in the same directory as your Flask app.

Step 3: Adjust Phase 2 Code (Flask App)

Your Phase 2 code should now be able to load the model without errors.

Adjusted Flask App (`app.py`):

python

Copy code

```python
from flask import Flask, request, jsonify
import numpy as np
from tensorflow.keras.models import load_model
import tensorflow as tf  # Ensure consistent TensorFlow imports

# Load the trained model
def load_trained_model(model_path='lstm_model.h5'):
    return load_model(model_path)

# Initialize Flask app
app = Flask(__name__)
model = load_trained_model()

@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Parse input data
        data = request.get_json(force=True)
        input_data = data['input']  # Expecting a 2D list (time_steps x num_features)

        # Convert input data to numpy array
        prediction_input = np.array(input_data)

        # Reshape input to match the model's expected input shape
        # The model expects input shape: (samples, time_steps, num_features)
        prediction_input = prediction_input.reshape((1, prediction_input.shape[0], prediction_input.shape[1]))

        # Make prediction
        prediction = model.predict(prediction_input)

        # Return the prediction as a list
        return jsonify({'prediction': prediction.tolist()})

    except Exception as e:
        return jsonify({'error': str(e)})

if __name__ == '__main__':
    # Run the Flask app
    app.run(host='0.0.0.0', port=5000)
```

Step 4: Provide All Necessary Code and Files

a. `requirements.txt`

Ensure that your `requirements.txt` includes all necessary dependencies:

makefile

Copy code

```makefile
Flask==2.0.1
tensorflow==2.17.0
numpy==1.26.4
```

b. `Dockerfile`

Your Dockerfile should be adjusted to match the updated Flask app.

dockerfile

Copy code

```dockerfile
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory inside the container
WORKDIR /app
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
# Copy the requirements file into the container
COPY requirements.txt /app/

# Install any required packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code into the container
COPY . /app

# Expose the port that the Flask app will run on
EXPOSE 5000

# Command to run the application
CMD ["python", "app.py"]
```

c. Build and Test Docker Image

1. **Build the Docker Image:**

   bash
   Copy code
   ```
   docker build -t your-dockerhub-username/traffic-prediction-app:latest .
   ```

2. **Run the Docker Container Locally:**

   bash
   Copy code
   ```
   docker run -p 5000:5000 your-dockerhub-username/traffic-prediction-app:latest
   ```

3. **Test the API Endpoint:**

   Send a test POST request to ensure the app works inside the container.

---

d. Push Docker Image to DockerHub

1. **Log in to DockerHub:**

   bash
   Copy code
   ```
   docker login
   ```

2. **Push the Image:**

   bash
   Copy code
   ```
   docker push your-dockerhub-username/traffic-prediction-app:latest
   ```

3. **Provide DockerHub Repository Link:**

   Create a file named `docker_image_link.txt` containing the link to your DockerHub repository.

---

e. Kubernetes Deployment Files

`deployment.yaml`:

yaml
Copy code
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: traffic-prediction-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: traffic-prediction
  template:
    metadata:
      labels:
        app: traffic-prediction
    spec:
      containers:
      - name: traffic-prediction-container
        image: your-dockerhub-username/traffic-prediction-app:latest
        ports:
        - containerPort: 5000
```

`service.yaml`:

yaml
Copy code
```
apiVersion: v1
kind: Service
metadata:
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
  name: traffic-prediction-service
spec:
  selector:
    app: traffic-prediction
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5000
  type: LoadBalancer
```

Zip the YAML files:

- Place both YAML files in a directory.
- Zip them into `kubernetes_deployment_files.zip`.

---

f. API Documentation

Create `api_documentation.pdf` with detailed instructions on how to interact with your API.

Content of `api_documentation.pdf`:

Traffic Prediction API Documentation

Endpoint: `/predict`

- **URL:** `/predict`
- **Method:** `POST`
- **Content-Type:** `application/json`
- **Description:** Predicts traffic speed based on input data.

Request Body

- **Parameter:** `input`
- **Type:** List of lists (2D array)
- **Description:** Time series data for prediction. Should be shaped as `(time_steps, num_features)`.

Example Request:

json

Copy code

```json
{
  "input": [
    [0.1, 0.2, ..., 0.207],
    [0.2, 0.3, ..., 0.208],
    ... (10 time steps in total)
  ]
}
```

- The example assumes `time_steps=10` and `num_features=207`.

Response

- **Success Response:**
  - **Code:** `200 OK`
  - **Content:**

    json

    Copy code

    ```json
    {
      "prediction": [[0.5]]
    }
    ```

- **Error Response:**
  - **Code:** `400 Bad Request`
  - **Content:**

    json

    Copy code

    ```json
    {
      "error": "Error message detailing what went wrong."
    }
    ```

Sample Request Using `curl`:

bash

Copy code

```bash
curl -X POST -H "Content-Type: application/json" \
-d '{"input": [[0.1, 0.2, ..., 0.207], ..., [0.1, 0.2, ..., 0.207]]}' \
http://your-service-url/predict
```

Note: Replace `your-service-url` with the actual URL of your service.

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Step 5: Verify and Test Everything

- **Ensure that all files are in the same directory**:
    - `app.py`
    - `Dockerfile`
    - `requirements.txt`
    - `lstm_model.h5`
    - `deployment.yaml`
    - `service.yaml`
- **Test your Flask app locally**:

  bash
  Copy code
  ```
  python app.py
  ```
- **Test your Docker container locally**:

  bash
  Copy code
  ```
  docker run -p 5000:5000 your-dockerhub-username/traffic-prediction-app:latest
  ```
- **Deploy to Kubernetes**:

  bash
  Copy code
  ```
  kubectl apply -f deployment.yaml
  kubectl apply -f service.yaml
  ```
- **Check the status of pods and services**:

  bash
  Copy code
  ```
  kubectl get pods
  kubectl get services
  ```

Additional Notes

- **TensorFlow and Keras Versions**:
    - Ensure that the versions of TensorFlow and Keras used in Phase 1 and Phase 2 are consistent.
    - In both environments, use `tensorflow==2.17.0` as specified in `requirements.txt`.
- **Model Input Shape**:
    - The input data sent to the `/predict` endpoint must be a 2D list with shape `(10, 207)`.
- **Testing**:
    - Use actual numerical values in your test requests to ensure the model can make predictions.
- **Error Handling**:
    - The Flask app includes error handling to return a JSON response with an error message in case of exceptions.

Conclusion

By adjusting the way you specify loss functions and metrics when compiling your model, you prevent serialization and deserialization issues. This allows your Flask app to load the model without errors, ensuring your Phase 2 deployment works smoothly.

Let me know if you have any questions or need further assistance with any of the steps!

You said:

I am running into an issue with docker file. C:\foai\94879-starter-code-Team-Project\Dockerfile (OpAI_VM) C:\foai>docker build -t mtbogush/model-api:v1 . [+] Building 0.1s (1/1) FINISHED docker:desktop-linux => [internal] load build definition from Dockerfile 0.0s => => transferring dockerfile: 2B 0.0s ERROR: failed to solve: failed to read dockerfile: open Dockerfile: no such file or directory View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/zjglqfj4l80mp2yybz383etwq # Use an official Python runtime as a parent image FROM python:3.13-slim # Set the working directory inside the container WORKDIR /app # Copy the requirements file into the container at /app COPY requirements.txt /app/ # Install any required packages specified in requirements.txt RUN pip install --no-cache-dir -r requirements.txt # Copy the rest of the application code into the container COPY . /app # Expose the port that the Flask app will run on EXPOSE 5000 # Command to run the application CMD ["python", "starter_notebook_phase_2.py"] from flask import Flask, request, jsonify import numpy

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

as np from tensorflow.keras.models import load_model import tensorflow as tf # Ensure consistent TensorFlow imports # Load the trained model def load_trained_model(model_path='lstm_model.h5'): return load_model(model_path) # Initialize Flask app app = Flask(__name__) model = load_trained_model() @app.route('/predict', methods=['POST']) def predict(): try: # Parse input data data = request.get_json(force=True) input_data = data['input'] # Expecting a 2D list (time_steps x num_features) # Convert input data to numpy array prediction_input = np.array(input_data) # Reshape input to match the model's expected input shape # The model expects input shape: (samples, time_steps, num_features) prediction_input = prediction_input.reshape((1, prediction_input.shape[0], prediction_input.shape[1])) # Make prediction prediction = model.predict(prediction_input) # Return the prediction as a list return jsonify({'prediction': prediction.tolist()}) except Exception as e: return jsonify({'error': str(e)}) if __name__ == '__main__': # Run the Flask app app.run(host='0.0.0.0', port=5000)

Thought for 26 seconds

It looks like the error you're encountering is because Docker can't find the `Dockerfile` in your current directory. Let's address this step by step.

Understanding the Error

The error message you received:

perl

Copy code

```
ERROR: failed to solve: failed to read dockerfile: open Dockerfile: no such file or directory
```

This indicates that Docker is trying to find a file named `Dockerfile` in your current directory (`C:\foai`), but it can't find it.

Solution

1. Ensure You're in the Correct Directory

Your `Dockerfile` is located at:

makefile

Copy code

```
C:\foai\94879-starter-code-Team-Project\Dockerfile
```

But you're running the `docker build` command from:

makefile

Copy code

```
C:\foai
```

Solution: Change your current directory to where the `Dockerfile` is located.

Command:

bash

Copy code

```
cd C:\foai\94879-starter-code-Team-Project\
```

2. Run the Docker Build Command from the Correct Directory

Once you're in the directory containing the `Dockerfile`, run:

bash

Copy code

```
docker build -t mtbogush/model-api:v1 .
```

- The `.` at the end specifies that Docker should use the current directory as the build context.
- Docker will now find the `Dockerfile` and proceed with the build.

Additional Recommendations

A. Verify the `Dockerfile`

Ensure that your `Dockerfile` is correctly named and not empty. It should be named exactly `Dockerfile` without any extensions (e.g., not `Dockerfile.txt`).

B. Update the Base Image in Your `Dockerfile`

You have:

dockerfile

Copy code

```
FROM python:3.13-slim
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

As of now, Python 3.13 may not be available. It's safer to use a stable version. Let's change it to Python 3.11 or 3.8.

Updated `Dockerfile`:

dockerfile

Copy code

```
# Use an official Python runtime as a parent image
FROM python:3.11-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the requirements file into the container
COPY requirements.txt /app/

# Install any required packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code into the container
COPY . /app

# Expose the port that the Flask app will run on
EXPOSE 5000

# Command to run the application
CMD ["python", "starter_notebook_phase_2.py"]
```

C. Ensure All Necessary Files Are Present

Make sure the following files are in the `C:\foai\94879-starter-code-Team-Project\` directory:

- `Dockerfile`
- `starter_notebook_phase_2.py` (your Flask app)
- `lstm_model.h5` (your trained model)
- `requirements.txt`

D. Check Your `requirements.txt`

Ensure your `requirements.txt` includes all necessary dependencies:

makefile

Copy code

```
Flask==2.0.1
tensorflow==2.17.0
numpy==1.26.4
```

E. Verify Your Flask App

Your `starter_notebook_phase_2.py` seems correctly set up. Just ensure it's in the same directory as your `Dockerfile`.

Step-by-Step Guide

1. **Navigate to the Correct Directory:**

   bash

   Copy code

   ```
   cd C:\foai\94879-starter-code-Team-Project\
   ```

2. **Build the Docker Image:**

   bash

   Copy code

   ```
   docker build -t mtbogush/model-api:v1 .
   ```

3. **Run the Docker Container Locally to Test:**

   bash

   Copy code

   ```
   docker run -p 5000:5000 mtbogush/model-api:v1
   ```

4. **Test the API Endpoint:**

   Use `curl` or Postman to send a test POST request to `http://localhost:5000/predict`.

   Example `curl` Command:

   bash

   Copy code

   ```
   curl -X POST -H "Content-Type: application/json" -d '{"input": [[0.1, 0.2, ..., 0.207], ..., [0.1, 0.2, ..., 0.207]]}' http://localh
   ```

   Replace `...` with actual numerical values.

5. **Push the Docker Image to DockerHub:**

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

bash
Copy code

```
docker login
docker push mtbogush/model-api:v1
```

6. **Prepare Kubernetes Deployment Files:**

   Ensure your `deployment.yaml` and `service.yaml` are correctly set up, as previously provided.

Common Pitfalls and Solutions

1. Empty or Incorrect `Dockerfile`

Ensure your `Dockerfile` is not empty and has the correct syntax.

2. File Naming

- Ensure all file names are exact and case-sensitive.
- The `Dockerfile` must be named `Dockerfile` without any extension.

3. File Locations

- All files required for the Docker build should be in the same directory or properly referenced.

4. Permissions

- Ensure you have the necessary permissions to read the files and execute the Docker build command.

Summary

- **Error Cause:** Docker couldn't find the `Dockerfile` because it was not in the current directory.
- **Solution:** Change to the directory containing the `Dockerfile` or specify the path to the `Dockerfile` and build context.
- **Additional Steps:** Update the base image, ensure all necessary files are present, and verify file names and locations.

Next Steps

- **Test Thoroughly:** After building the Docker image, run it locally and test the API.
- **Push to DockerHub:** Once satisfied, push your image to DockerHub.
- **Deploy to Kubernetes:** Use your deployment files to deploy the application.
- **Document Everything:** Ensure all your steps and configurations are well-documented for submission.

Let me know if you need any further assistance or if you encounter any other issues!

You said:

What happened here?

```
(OpAI_VM) C:\foai\94879-starter-code-Team-Project>docker build -t mtbogush/model-api:v1 . [+] Building 23.2s (9/10) docker:desktop-
linux => [internal] load build definition from Dockerfile 0.0s => => transferring dockerfile: 608B 0.0s => [internal] load metadata for docker.io/library/python:3.13-
slim 1.0s => [auth] library/python:pull token for registry-1.docker.io 0.0s => [internal] load .dockerignore 0.0s => => transferring context: 2B 0.0s => [1/5] FROM
docker.io/library/python:3.13-slim@sha256:2ec5a4a5c3e919570f57675471f081d6299668d909feabd8d4803c6c61af666c 3.0s => => resolve
docker.io/library/python:3.13-slim@sha256:2ec5a4a5c3e919570f57675471f081d6299668d909feabd8d4803c6c61af666c 0.0s => =>
sha256:2ec5a4a5c3e919570f57675471f081d6299668d909feabd8d4803c6c61af666c 9.12kB / 9.12kB 0.0s => =>
sha256:54b732625bf1ae735c9f917e77bbc728a22ba079eb8faee40dfc6879f5a559c1 1.75kB / 1.75kB 0.0s => =>
sha256:7f865f7658c73a1ceb5ed30c941fc855c8f3f494b8d8736d91ea73c7c8a004c5 4.98kB / 4.98kB 0.0s => =>
sha256:adbbeb085bcc4c9cf6c339eded7800b0f6ed6e8a486907094b606548bdd89c89 3.51MB / 3.51MB 0.3s => =>
sha256:f7fde2f40da07c0c30a52dcfde42aab768dc568eb43984d9ab8b765f87235440 12.32MB / 12.32MB 0.7s => =>
sha256:46bb254b894ea1bcec6f60f1601c32af90a460ca137682d79e1f5a33daeffd10 251B / 251B 0.1s => => extracting
sha256:adbbeb085bcc4c9cf6c339eded7800b0f6ed6e8a486907094b606548bdd89c89 0.5s => => extracting
sha256:f7fde2f40da07c0c30a52dcfde42aab768dc568eb43984d9ab8b765f87235440 1.9s => => extracting
sha256:46bb254b894ea1bcec6f60f1601c32af90a460ca137682d79e1f5a33daeffd10 0.0s => [internal] load build context 0.0s => => transferring context: 13.60kB
0.0s => [2/5] WORKDIR /app 0.1s => [3/5] COPY requirements.txt /app/ 0.1s => ERROR [4/5] RUN pip install --no-cache-dir -r requirements.txt 18.9s ------ >
[4/5] RUN pip install --no-cache-dir -r requirements.txt: 2.990 Collecting Flask==2.0.1 (from -r requirements.txt (line 1)) 3.138 Downloading Flask-2.0.1-py3-none-
any.whl.metadata (3.8 kB) 3.448 Collecting numpy==1.26.4 (from -r requirements.txt (line 2)) 3.477 Downloading numpy-1.26.4.tar.gz (15.8 MB) 4.125
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.8/15.8 MB 26.5 MB/s eta 0:00:00 11.32 Installing build dependencies: started 15.59 Installing build
dependencies: finished with status 'done' 15.59 Getting requirements to build wheel: started 15.75 Getting requirements to build wheel: finished with status 'done'
15.75 Installing backend dependencies: started 16.95 Installing backend dependencies: finished with status 'done' 16.96 Preparing metadata (pyproject.toml): started
18.31 Preparing metadata (pyproject.toml): finished with status 'error' 18.32 error: subprocess-exited-with-error 18.32 18.32 × Preparing metadata (pyproject.toml)
did not run successfully. 18.32 │ exit code: 1 18.32 ╰─> [20 lines of output] 18.32 + /usr/local/bin/python3.13 /tmp/pip-install-
twnrf13z/numpy_9ed9e94184394c48ab6e0e057468f0a5/vendored-meson/meson/meson.py setup /tmp/pip-install-
twnrf13z/numpy_9ed9e94184394c48ab6e0e057468f0a5 /tmp/pip-install-twnrf13z/numpy_9ed9e94184394c48ab6e0e057468f0a5/.mesonpy-_7womps2 -
Dbuildtype=release -Db_ndebug=if-release -Db_vscrt=md --native-file=/tmp/pip-install-twnrf13z/numpy_9ed9e94184394c48ab6e0e057468f0a5/.mesonpy-
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

_7womps2/meson-python-native-file.ini 18.32 The Meson build system 18.32 Version: 1.2.99 18.32 Source dir: /tmp/pip-install-twnrf13z/numpy_9ed9e94184394c48ab6e0e057468f0a5 18.32 Build dir: /tmp/pip-install-twnrf13z/numpy_9ed9e94184394c48ab6e0e057468f0a5/.mesonpy-_7womps2 18.32 Build type: native build 18.32 Project name: NumPy 18.32 Project version: 1.26.4 18.32 18.32 ../meson.build:1:0: ERROR: Unknown compiler(s): [['cc'], ['gcc'], ['clang'], ['nvc'], ['pgcc'], ['icc'], ['icx']] 18.32 The following exception(s) were encountered: 18.32 Running `cc --version` gave "[Errno 2] No such file or directory: 'cc'" 18.32 Running `gcc --version` gave "[Errno 2] No such file or directory: 'gcc'" 18.32 Running `clang --version` gave "[Errno 2] No such file or directory: 'clang'" 18.32 Running `nvc --version` gave "[Errno 2] No such file or directory: 'nvc'" 18.32 Running `pgcc --version` gave "[Errno 2] No such file or directory: 'pgcc'" 18.32 Running `icc --version` gave "[Errno 2] No such file or directory: 'icc'" 18.32 Running `icx --version` gave "[Errno 2] No such file or directory: 'icx'" 18.32 18.32 A full log can be found at /tmp/pip-install-twnrf13z/numpy_9ed9e94184394c48ab6e0e057468f0a5/.mesonpy-_7womps2/meson-logs/meson-log.txt 18.32 [end of output] 18.32 18.32 note: This error originates from a subprocess, and is likely not a problem with pip. 18.54 error: metadata-generation-failed 18.54 18.54 × Encountered error while generating package metadata. 18.54 ╰─> See above for output. 18.54 18.54 note: This is an issue with the package mentioned above, not pip. 18.54 hint: See above for details. ------ Dockerfile:11 -------------------- 9 | 10 | # Install any required packages specified in requirements.txt 11 | >>> RUN pip install --no-cache-dir -r requirements.txt 12 | 13 | # Copy the rest of the application code into the container -------------------- ERROR: failed to solve: process "/bin/sh -c pip install --no-cache-dir -r requirements.txt" did not complete successfully: exit code: 1 View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/tpgq0fubblvb79an5fig0px89

Thought for 12 seconds

I understand you're encountering an error while building your Docker image. Let's analyze the error, understand what's causing it, and provide a solution to fix it.

Understanding the Error

The error message you're seeing during the Docker build is:

vbnet

Copy code

```
ERROR: failed to solve: process "/bin/sh -c pip install --no-cache-dir -r requirements.txt" did not complete successfully: exit code: 1
```

Further up in the logs, the key part is:

css

Copy code

```
ERROR: Unknown compiler(s): [['cc'], ['gcc'], ['clang'], ['nvc'], ['pgcc'], ['icc'], ['icx']]
```

This indicates that during the installation of `numpy`, it needs to compile some code, but no C compiler is available in the Docker image.

Additionally, you're using:

dockerfile

Copy code

```
FROM python:3.13-slim
```

Issues Identified:

1. **Python 3.13 Not Officially Released:**
   - As of my knowledge cutoff in September 2021, Python 3.13 is not officially released. Even if it were, many packages may not yet support it, leading to compatibility issues.
2. **Missing Build Tools in `slim` Image:**
   - The `slim` variants of Docker images are minimal and do not include build tools like `gcc`, which are necessary for compiling packages from source.
3. **`numpy` Version Compatibility:**
   - `numpy==1.26.4` may not have pre-built wheels for the Python version you're using, causing `pip` to attempt to build it from source.

Solution

1. Change the Base Image to a Stable Python Version

Let's use a stable Python version that's widely supported, such as Python 3.8 or Python 3.11.

Updated `Dockerfile` Base Image:

dockerfile

Copy code

```
FROM python:3.11-slim
```

2. Install Necessary Build Tools

Since you're using a `slim` image, you need to install build tools required to compile packages like `numpy`.

Add the Following to Your `Dockerfile`:

dockerfile

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Copy code

```
# Install build tools
RUN apt-get update && apt-get install -y build-essential
```

Alternatively, if you want to minimize the size of your Docker image, you can try to use pre-built wheels of `numpy` compatible with your Python version.

3. Adjust `numpy` Version in `requirements.txt`

To avoid building `numpy` from source, specify a version that has pre-built wheels for your Python version.

Modify `requirements.txt`:

- Change `numpy==1.26.4` to a stable version like `numpy==1.25.2`.

Updated `requirements.txt`:

makefile

Copy code

```
Flask==2.0.1
numpy==1.25.2
tensorflow==2.12.0
```

Note: Ensure that the versions of `tensorflow` and `numpy` are compatible.

Updated Files

1. Updated `Dockerfile`

dockerfile

Copy code

```
# Use an official Python runtime as a parent image
FROM python:3.11-slim

# Set the working directory inside the container
WORKDIR /app

# Install build tools
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

# Copy the requirements file into the container
COPY requirements.txt /app/

# Install any required packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code into the container
COPY . /app

# Expose the port that the Flask app will run on
EXPOSE 5000

# Command to run the application
CMD ["python", "starter_notebook_phase_2.py"]
```

Explanation:

- **Install Build Tools:**

  dockerfile

  Copy code

  ```
  RUN apt-get update && apt-get install -y --no-install-recommends \
      build-essential \
      && rm -rf /var/lib/apt/lists/*
  ```

  - This installs `gcc` and other essential tools needed to compile packages from source.

- **Clean Up:**
  - `rm -rf /var/lib/apt/lists/*` cleans up the apt cache to reduce the image size.

2. Updated `requirements.txt`

text

Copy code

```
Flask==2.0.1
numpy==1.25.2
tensorflow==2.12.0
```

Notes:

- **TensorFlow Version:**
  - `tensorflow==2.12.0` is compatible with Python 3.11 and `numpy==1.25.2`.

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

  - Adjust if necessary based on compatibility and availability.

Step-by-Step Guide

1. Update Your `Dockerfile` and `requirements.txt`

Make sure your `Dockerfile` and `requirements.txt` reflect the updates provided above.

2. Build the Docker Image

Run the Docker build command from the directory containing your `Dockerfile`:

bash
Copy code

```
cd C:\foai\94879-starter-code-Team-Project\
docker build -t mtbogush/model-api:v1 .
```

Note: Ensure that the `Dockerfile`, `starter_notebook_phase_2.py`, `lstm_model.h5`, and `requirements.txt` are all in the current directory.

3. Test the Docker Image Locally

Run the Docker container:

bash
Copy code

```
docker run -p 5000:5000 mtbogush/model-api:v1
```

4. Test the API Endpoint

Use `curl` or Postman to send a test POST request:

bash
Copy code

```
curl -X POST -H "Content-Type: application/json" \
-d '{"input": [[0.1, 0.2, ..., 0.207], ..., [0.1, 0.2, ..., 0.207]]}' \
http://localhost:5000/predict
```

Replace `...` with actual numerical values.

5. Push the Docker Image to DockerHub

bash
Copy code

```
docker login
docker push mtbogush/model-api:v1
```

Additional Tips

1. Ensure Package Compatibility

Verify that the versions of `tensorflow` and `numpy` in `requirements.txt` are compatible.

- You can check the compatibility matrix in the TensorFlow documentation or release notes.

2. Use Pre-built Wheels

By specifying versions of packages that have pre-built wheels for your Python version, you avoid the need to compile from source.

3. Monitor Image Size

Installing build tools increases the size of your Docker image. If image size is a concern, consider using a non-slim base image or multi-stage builds.

4. Check Python Version Compatibility

Ensure that all your dependencies support the Python version specified in your Docker image.

- If in doubt, use a widely supported Python version like 3.8.

5. Clean Up After Installation

To keep your Docker image slim, remove unnecessary files and caches after installations.

Alternative Solution: Use a Non-slim Base Image

If you prefer not to install build tools manually, you can use the full Python image, which includes build tools.

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Updated `Dockerfile` Using Non-slim Image:

dockerfile

Copy code

```dockerfile
# Use an official Python runtime as a parent image
FROM python:3.11

# Set the working directory inside the container
WORKDIR /app

# Copy the requirements file into the container
COPY requirements.txt /app/

# Install any required packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code into the container
COPY . /app

# Expose the port that the Flask app will run on
EXPOSE 5000

# Command to run the application
CMD ["python", "starter_notebook_phase_2.py"]
```

Pros:

- Build tools are already included.
- Simpler Dockerfile.

Cons:

- Larger image size.

Summary

- **Issue:** Docker build fails because of missing build tools when compiling `numpy` from source in a `slim` image.
- **Solution:** Install build tools in the `Dockerfile` or use a non-slim base image.
- **Adjustments:**
  - Change the base image to a stable Python version (e.g., `python:3.11-slim`).
  - Install `build-essential` tools.
  - Adjust package versions in `requirements.txt` to ensure compatibility.

Next Steps

1. **Implement the Suggested Changes:**
   - Update your `Dockerfile` and `requirements.txt` as per the guidance.
2. **Rebuild and Test the Docker Image:**
   - Run `docker build` and ensure the build completes successfully.
   - Test the container locally to verify that the API works.
3. **Push the Image to DockerHub:**
   - Once satisfied, push the image to your DockerHub repository.
4. **Proceed with Kubernetes Deployment:**
   - Use your `deployment.yaml` and `service.yaml` files to deploy your application.
   - Verify that the application is running and accessible.

Let Me Know If You Need Further Assistance

If you encounter any other issues or have questions about any of the steps, please don't hesitate to ask. I'm here to help you successfully complete Phase 2 of your project.

You said:

I keep running into this RUN pip install --no-cache-dir -r requirements.txt in docker file. ERROR: 58.44 cpdef _sample_without_replacement_with_pool(np.int_t n_population, 58.44 np.int_t n_samples, 58.44 ^ 58.44 -- 58.44 58.44 sklearn/utils/_random.pyx:106:44: 'int_t' is not a type identifier 58.44 58.44 Error compiling Cython file: 58.44 -- 58.44 ... 58.44 58.44 return out 58.44 58.44 58.44 cpdef _sample_without_replacement_with_reservoir_sampling(58.44 np.int_t n_population, 58.44 ^ 58.44 -- 58.44 58.44 sklearn/utils/_random.pyx:165:4: 'int_t' is not a type identifier 58.44 58.44 Error compiling Cython file: 58.44 -- 58.44 ... 58.44 return out 58.44 58.44 58.44 cpdef _sample_without_replacement_with_reservoir_sampling(58.44 np.int_t n_population, 58.44 np.int_t n_samples, 58.44 ^ 58.44 -- 58.44 58.44 sklearn/utils/_random.pyx:166:4: 'int_t' is not a type identifier 58.44 58.44

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
Error compiling Cython file: 58.44 ------------------------------------------------------------ 58.44 ... 58.44 out[j] = i 58.44 58.44 return out 58.44 58.44 58.44 cpdef
sample_without_replacement(np.int_t n_population, 58.44 ^ 58.44 ------------------------------------------------------------ 58.44 58.44 sklearn/utils/_random.pyx:223:33:
'int_t' is not a type identifier 58.44 58.44 Error compiling Cython file: 58.44 ------------------------------------------------------------ 58.44 ... 58.44 58.44 return out 58.44
58.44 58.44 cpdef sample_without_replacement(np.int_t n_population, 58.44 np.int_t n_samples, 58.44 ^ 58.44 ------------------------------------------------------------
58.44 58.44 sklearn/utils/_random.pyx:224:33: 'int_t' is not a type identifier 58.44 58.44 Error compiling Cython file: 58.44 -------------------------------------------------
----------- 58.44 ... 58.44 out : ndarray of shape (n_samples,) 58.44 The sampled subsets of integer. 58.44 """ 58.44
_sample_without_replacement_check_input(n_population, n_samples) 58.44 58.44 cdef np.int_t i 58.44 ^ 58.44 ------------------------------------------------------------
58.44 58.44 sklearn/utils/_random.pyx:84:9: 'int_t' is not a type identifier 58.44 58.44 Error compiling Cython file: 58.44 ----------------------------------------------------
-------- 58.44 ... 58.44 The sampled subsets of integer. 58.44 """ 58.44 _sample_without_replacement_check_input(n_population, n_samples) 58.44 58.44 cdef
np.int_t i 58.44 cdef np.int_t j 58.44 ^ 58.44 ------------------------------------------------------------ 58.44 58.44 sklearn/utils/_random.pyx:85:9: 'int_t' is not a type
identifier 58.44 58.44 Error compiling Cython file: 58.44 ------------------------------------------------------------ 58.44 ... 58.44 """ 58.44
_sample_without_replacement_check_input(n_population, n_samples) 58.44 58.44 cdef np.int_t i 58.44 cdef np.int_t j 58.44 cdef np.ndarray[np.int_t, ndim=1] out =
np.empty((n_samples, ), dtype=int) 58.44 ^ 58.44 ------------------------------------------------------------ 58.44 58.44 sklearn/utils/_random.pyx:86:23: Invalid type.
58.44 Traceback (most recent call last): 58.44 File "/tmp/pip-build-env-unvmmobj/overlay/lib/python3.11/site-packages/Cython/Build/Dependencies.py", line 1345,
in cythonize_one_helper 58.44 return cythonize_one(*m) 58.44 ^^^^^^^^^^^^^^^^^ 58.44 File "/tmp/pip-build-env-unvmmobj/overlay/lib/python3.11/site-
packages/Cython/Build/Dependencies.py", line 1321, in cythonize_one 58.44 raise CompileError(None, pyx_file) 58.44 Cython.Compiler.Errors.CompileError:
sklearn/utils/_random.pyx 58.44 58.44 Error compiling Cython file: 58.44 ------------------------------------------------------------ 58.44 ... 58.44 # Max value for our
rand_r replacement (near the bottom). 58.44 # We don't use RAND_MAX because it's different across platforms and 58.44 # particularly tiny on Windows/MSVC.
58.44 RAND_R_MAX = 0x7FFFFFFF 58.44 58.44 cpdef sample_without_replacement(np.int_t n_population, 58.44 ^ 58.44 ------------------------------------------------
------------ 58.44 58.44 sklearn/utils/_random.pxd:18:33: 'int_t' is not a type identifier 58.44 58.44 Error compiling Cython file: 58.44 --------------------------------------
---------------------- 58.44 ... 58.44 # We don't use RAND_MAX because it's different across platforms and 58.44 # particularly tiny on Windows/MSVC. 58.44
RAND_R_MAX = 0x7FFFFFFF 58.44 58.44 cpdef sample_without_replacement(np.int_t n_population, 58.44 np.int_t n_samples, 58.44 ^ 58.44 ------------------------
------------------------------------ 58.44 58.44 sklearn/utils/_random.pxd:19:33: 'int_t' is not a type identifier 58.44 Traceback (most recent call last): 58.44 File
"/tmp/pip-build-env-unvmmobj/overlay/lib/python3.11/site-packages/Cython/Build/Dependencies.py", line 1345, in cythonize_one_helper 58.44 return
cythonize_one(*m) 58.44 ^^^^^^^^^^^^^^^^^ 58.44 File "/tmp/pip-build-env-unvmmobj/overlay/lib/python3.11/site-packages/Cython/Build/Dependencies.py", line
1321, in cythonize_one 58.44 raise CompileError(None, pyx_file) 58.44 Cython.Compiler.Errors.CompileError: sklearn/utils/_seq_dataset.pyx 58.44 performance
hint: sklearn/utils/_weight_vector.pyx:169:29: Exception check after calling 'reset_wscale' will always require the GIL to be acquired. 58.44 Possible solutions:
58.44 1. Declare 'reset_wscale' as 'noexcept' if you control the definition and you're sure you don't want the function to raise exceptions. 58.44 2. Use an 'int' return
type on 'reset_wscale' to allow an error code to be returned. 58.44 performance hint: sklearn/utils/_weight_vector.pyx:174:17: Exception check after calling
'__pyx_fuse_1_axpy' will always require the GIL to be acquired. 58.44 Possible solutions: 58.44 1. Declare '__pyx_fuse_1_axpy' as 'noexcept' if you control the
definition and you're sure you don't want the function to raise exceptions. 58.44 2. Use an 'int' return type on '__pyx_fuse_1_axpy' to allow an error code to be
returned. 58.44 performance hint: sklearn/utils/_weight_vector.pyx:176:17: Exception check after calling '__pyx_fuse_1_scal' will always require the GIL to be
acquired. 58.44 Possible solutions: 58.44 1. Declare '__pyx_fuse_1_scal' as 'noexcept' if you control the definition and you're sure you don't want the function to
raise exceptions. 58.44 2. Use an 'int' return type on '__pyx_fuse_1_scal' to allow an error code to be returned. 58.44 performance hint:
sklearn/utils/_weight_vector.pyx:180:13: Exception check after calling '__pyx_fuse_1_scal' will always require the GIL to be acquired. 58.44 Possible solutions:
58.44 1. Declare '__pyx_fuse_1_scal' as 'noexcept' if you control the definition and you're sure you don't want the function to raise exceptions. 58.44 2. Use an 'int'
return type on '__pyx_fuse_1_scal' to allow an error code to be returned. 58.44 multiprocessing.pool.RemoteTraceback: 58.44 """ 58.44 Traceback (most recent call
last): 58.44 File "/usr/local/lib/python3.11/multiprocessing/pool.py", line 125, in worker 58.44 result = (True, func(*args, **kwds)) 58.44 ^^^^^^^^^^^^^^^^^^^
58.44 File "/usr/local/lib/python3.11/multiprocessing/pool.py", line 48, in mapstar 58.44 return list(map(*args)) 58.44 ^^^^^^^^^^^^^^^^ 58.44 File "/tmp/pip-build-
env-unvmmobj/overlay/lib/python3.11/site-packages/Cython/Build/Dependencies.py", line 1345, in cythonize_one_helper 58.44 return cythonize_one(*m) 58.44
^^^^^^^^^^^^^^^^^ 58.44 File "/tmp/pip-build-env-unvmmobj/overlay/lib/python3.11/site-packages/Cython/Build/Dependencies.py", line 1321, in cythonize_one
58.44 raise CompileError(None, pyx_file) 58.44 Cython.Compiler.Errors.CompileError: sklearn/ensemble/_hist_gradient_boosting/splitting.pyx 58.44 """ 58.44
58.44 The above exception was the direct cause of the following exception: 58.44 58.44 Traceback (most recent call last): 58.44 File "/usr/local/lib/python3.11/site-
packages/pip/_vendor/pyproject_hooks/_in_process/_in_process.py", line 353, in <module> 58.44 [ 1/53] Cythonizing sklearn/__check_build/_check_build.pyx
58.44 [ 2/53] Cythonizing sklearn/_isotonic.pyx 58.44 [ 3/53] Cythonizing sklearn/cluster/_dbscan_inner.pyx 58.44 [ 4/53] Cythonizing
sklearn/cluster/_hierarchical_fast.pyx 58.44 [ 5/53] Cythonizing sklearn/cluster/_k_means_elkan.pyx 58.44 [ 6/53] Cythonizing sklearn/cluster/_k_means_fast.pyx
58.44 [ 7/53] Cythonizing sklearn/cluster/_k_means_lloyd.pyx 58.44 [ 8/53] Cythonizing sklearn/datasets/_svmlight_format_fast.pyx 58.44 [ 9/53] Cythonizing
sklearn/decomposition/_cdnmf_fast.pyx 58.44 [10/53] Cythonizing sklearn/decomposition/_online_lda_fast.pyx 58.44 [11/53] Cythonizing
sklearn/ensemble/_gradient_boosting.pyx 58.44 [12/53] Cythonizing sklearn/ensemble/_hist_gradient_boosting/_binning.pyx 58.44 [13/53] Cythonizing
sklearn/ensemble/_hist_gradient_boosting/_bitset.pyx 58.44 [14/53] Cythonizing sklearn/ensemble/_hist_gradient_boosting/_gradient_boosting.pyx 58.44 [15/53]
Cythonizing sklearn/ensemble/_hist_gradient_boosting/_loss.pyx 58.44 [16/53] Cythonizing sklearn/ensemble/_hist_gradient_boosting/_predictor.pyx 58.44 [17/53]
Cythonizing sklearn/ensemble/_hist_gradient_boosting/common.pyx 58.44 [18/53] Cythonizing sklearn/ensemble/_hist_gradient_boosting/histogram.pyx 58.44
[19/53] Cythonizing sklearn/ensemble/_hist_gradient_boosting/splitting.pyx 58.44 [20/53] Cythonizing sklearn/ensemble/_hist_gradient_boosting/utils.pyx 58.44
[21/53] Cythonizing sklearn/feature_extraction/_hashing_fast.pyx 58.44 [22/53] Cythonizing sklearn/linear_model/_cd_fast.pyx 58.44 [23/53] Cythonizing
sklearn/linear_model/_sag_fast.pyx 58.44 [24/53] Cythonizing sklearn/linear_model/_sgd_fast.pyx 58.44 [25/53] Cythonizing
sklearn/manifold/_barnes_hut_tsne.pyx 58.44 [26/53] Cythonizing sklearn/manifold/_utils.pyx 58.44 [27/53] Cythonizing sklearn/metrics/_pairwise_fast.pyx 58.44
[28/53] Cythonizing sklearn/metrics/cluster/_expected_mutual_info_fast.pyx 58.44 [29/53] Cythonizing sklearn/neighbors/_ball_tree.pyx 58.44 [30/53] Cythonizing
sklearn/neighbors/_dist_metrics.pyx 58.44 [31/53] Cythonizing sklearn/neighbors/_kd_tree.pyx 58.44 [32/53] Cythonizing sklearn/neighbors/_quad_tree.pyx 58.44
[33/53] Cythonizing sklearn/neighbors/_typedefs.pyx 58.44 [34/53] Cythonizing sklearn/preprocessing/_csr_polynomial_expansion.pyx 58.44 [35/53] Cythonizing
sklearn/svm/_liblinear.pyx 58.44 [36/53] Cythonizing sklearn/svm/_libsvm.pyx 58.44 [37/53] Cythonizing sklearn/svm/_libsvm_sparse.pyx 58.44 [38/53]
Cythonizing sklearn/svm/_newrand.pyx 58.44 [39/53] Cythonizing sklearn/tree/_criterion.pyx 58.44 [40/53] Cythonizing sklearn/tree/_splitter.pyx 58.44 [41/53]
Cythonizing sklearn/tree/_tree.pyx 58.44 [42/53] Cythonizing sklearn/tree/_utils.pyx 58.44 [43/53] Cythonizing sklearn/utils/_cython_blas.pyx 58.44 [44/53]
Cythonizing sklearn/utils/_fast_dict.pyx 58.44 [45/53] Cythonizing sklearn/utils/_logistic_sigmoid.pyx 58.44 [46/53] Cythonizing
sklearn/utils/_openmp_helpers.pyx 58.44 [47/53] Cythonizing sklearn/utils/_random.pyx 58.44 [48/53] Cythonizing sklearn/utils/_seq_dataset.pyx 58.44 [49/53]
Cythonizing sklearn/utils/_weight_vector.pyx 58.44 [50/53] Cythonizing sklearn/utils/arrayfuncs.pyx 58.44 [51/53] Cythonizing
sklearn/utils/graph_shortest_path.pyx 58.44 [52/53] Cythonizing sklearn/utils/murmurhash.pyx 58.44 [53/53] Cythonizing sklearn/utils/sparsefuncs_fast.pyx 58.44
main() 58.44 File "/usr/local/lib/python3.11/site-packages/pip/_vendor/pyproject_hooks/_in_process/_in_process.py", line 335, in main 58.44 json_out['return_val']
= hook(**hook_input['kwargs']) 58.44 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ 58.44 File "/usr/local/lib/python3.11/site-
packages/pip/_vendor/pyproject_hooks/_in_process/_in_process.py", line 149, in prepare_metadata_for_build_wheel 58.44 return hook(metadata_directory,
config_settings) 58.44 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ 58.44 File "/tmp/pip-build-env-unvmmobj/overlay/lib/python3.11/site-
packages/setuptools/build_meta.py", line 373, in prepare_metadata_for_build_wheel 58.44 self.run_setup() 58.44 File "/tmp/pip-build-env-
unvmmobj/overlay/lib/python3.11/site-packages/setuptools/build_meta.py", line 503, in run_setup 58.44 super().run_setup(setup_script=setup_script) 58.44 File
"/tmp/pip-build-env-unvmmobj/overlay/lib/python3.11/site-packages/setuptools/build_meta.py", line 318, in run_setup 58.44 exec(code, locals()) 58.44 File "
<string>", line 301, in <module> 58.44 File "<string>", line 297, in setup_package 58.44 File "/tmp/pip-build-env-unvmmobj/overlay/lib/python3.11/site-
packages/numpy/distutils/core.py", line 135, in setup 58.44 config = configuration() 58.44 ^^^^^^^^^^^^^^^ 58.44 File "<string>", line 188, in configuration 58.44
File "/tmp/pip-build-env-unvmmobj/overlay/lib/python3.11/site-packages/numpy/distutils/misc_util.py", line 1041, in add_subpackage 58.44 config_list =
self.get_subpackage(subpackage_name, subpackage_path, 58.44 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ 58.44 File "/tmp/pip-build-env-
unvmmobj/overlay/lib/python3.11/site-packages/numpy/distutils/misc_util.py", line 1007, in get_subpackage 58.44 config =
self._get_configuration_from_setup_py( 58.44 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ 58.44 File "/tmp/pip-build-env-unvmmobj/overlay/lib/python3.11/site-
packages/numpy/distutils/misc_util.py", line 949, in _get_configuration_from_setup_py 58.44 config = setup_module.configuration(*args) 58.44
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ 58.44 File "/tmp/pip-install-mcvxp08w/scikit-learn_312a456a0f734f0a8fb3a510585691dc/sklearn/setup.py", line 83, in
configuration 58.44 cythonize_extensions(top_path, config) 58.44 File "/tmp/pip-install-mcvxp08w/scikit-
learn_312a456a0f734f0a8fb3a510585691dc/sklearn/_build_utils/__init__.py", line 70, in cythonize_extensions 58.44 config.ext_modules = cythonize( 58.44
^^^^^^^^^^ 58.44 File "/tmp/pip-build-env-unvmmobj/overlay/lib/python3.11/site-packages/Cython/Build/Dependencies.py", line 1145, in cythonize 58.44
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
result.get(99999) # seconds 58.44 ^^^^^^^^^^^^^^^^^^ 58.44 File "/usr/local/lib/python3.11/multiprocessing/pool.py", line 774, in get 58.44 raise self._value 58.44
Cython.Compiler.Errors.CompileError: sklearn/ensemble/_hist_gradient_boosting/splitting.pyx 58.44 [end of output] 58.44 58.44 note: This error originates from a
subprocess, and is likely not a problem with pip. 58.44 error: metadata-generation-failed 58.44 58.44 × Encountered error while generating package metadata. 58.44
╰─> See above for output. 58.44 58.44 note: This is an issue with the package mentioned above, not pip. 58.44 hint: See above for details. 58.63 58.63 [notice] A
new release of pip is available: 24.0 -> 24.2 58.63 [notice] To update, run: pip install --upgrade pip ------ Dockerfile:20 -------------------- 18 | 19 | # Install any
required packages specified in requirements.txt 20 | >>> RUN pip install --no-cache-dir -r requirements.txt 21 | 22 | # Copy the rest of the application code into the
container -------------------- ERROR: failed to solve: process "/bin/sh -c pip install --no-cache-dir -r requirements.txt" did not complete successfully: exit code: 1
View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/4zhvigeol5zrdjsa6dmjucq6b
```

Thought for 20 seconds

I'm sorry to hear you're encountering issues during the Docker build process. The error you're facing is related to compiling `scikit-learn` and is common when there are compatibility issues between package versions, Python versions, and the need for additional build tools.

Let's address this step by step to resolve the error and help you successfully build your Docker image.

Understanding the Error

The key parts of the error message are:

- **Cython Compilation Errors**: Multiple errors like `'int_t' is not a type identifier` indicate issues during the Cython compilation of `scikit-learn` components.
- **`CompileError: sklearn/...`**: Errors pointing to specific Cython files in `scikit-learn`.
- **Missing Build Tools**: The `slim` variants of Python Docker images lack build tools necessary for compiling packages from source.
- **Python Version Compatibility**: Using Python 3.13 (or any unreleased version) can cause incompatibilities with packages that haven't been updated to support it.

Solution Overview

1. **Change the Base Image to a Stable Python Version**: Use a widely supported Python version like 3.9 or 3.8.
2. **Adjust `requirements.txt`**: Ensure all package versions are compatible with your Python version and each other.
3. **Install Necessary Build Tools**: Add required system packages to your Docker image.
4. **Install Cython Before Installing Packages That Depend on It**: Some packages like `scikit-learn` require Cython during installation.
5. **Verify Whether All Packages Are Necessary**: Remove unnecessary packages to simplify your build.

Detailed Steps

1. Change the Base Image to Python 3.9

Update your `Dockerfile` to use a stable Python version. Python 3.9 is widely supported and compatible with most packages.

Updated `Dockerfile` Base Image:

dockerfile

Copy code

```dockerfile
FROM python:3.9-slim
```

2. Adjust Your `requirements.txt`

Ensure your `requirements.txt` includes only the necessary packages and specifies versions compatible with Python 3.9.

Check Your Current `requirements.txt`:

- If `scikit-learn` is listed and not required for your Flask app, remove it.
- If you need `scikit-learn`, specify a compatible version.

Updated `requirements.txt`:

text

Copy code

```text
Flask==2.0.1
numpy==1.21.0
tensorflow==2.5.0
# Include scikit-learn only if necessary
# scikit-learn==0.24.2
```

3. Install Necessary Build Tools

Since `slim` images are minimal, you need to install build tools like `gcc` to compile packages.

Add to Your `Dockerfile`:

dockerfile

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Copy code

```
# Install build tools
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    && rm -rf /var/lib/apt/lists/*
```

4. Install Cython Before Other Packages

If you need to install packages like `scikit-learn` that require Cython, install Cython first.

Add to Your `requirements.txt` if Needed:

text

Copy code

```
Cython==0.29.23
```

Alternatively, Install Cython in the `Dockerfile`:

dockerfile

Copy code

```
# Install Cython
RUN pip install Cython==0.29.23
```

5. Update Your `Dockerfile`

Here's the complete updated `Dockerfile`:

dockerfile

Copy code

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Set the working directory inside the container
WORKDIR /app

# Install build tools
RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

# Install Cython if needed
RUN pip install Cython==0.29.23

# Copy the requirements file into the container
COPY requirements.txt /app/

# Install any required packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code into the container
COPY . /app

# Expose the port that the Flask app will run on
EXPOSE 5000

# Command to run the application
CMD ["python", "starter_notebook_phase_2.py"]
```

6. Rebuild Your Docker Image

Run the following commands:

bash

Copy code

```
# Navigate to your project directory
cd C:\foai\94879-starter-code-Team-Project\

# Build the Docker image
docker build -t mtbogush/model-api:v1 .
```

7. Test the Docker Image Locally

Run the Docker container:

bash

Copy code

```
docker run -p 5000:5000 mtbogush/model-api:v1
```

8. Test the API Endpoint

Use `curl` or Postman to send a test POST request to `http://localhost:5000/predict`.

Example `curl` Command:

bash

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Copy code

```
curl -X POST -H "Content-Type: application/json" \
-d '{"input": [[0.1, 0.2, ..., 0.207], ..., [0.1, 0.2, ..., 0.207]]}' \
http://localhost:5000/predict
```

Note: Replace `...` with actual numerical values.

9. Push the Docker Image to DockerHub

If everything works correctly:

bash
Copy code

```
docker login
docker push mtbogush/model-api:v1
```

Additional Considerations

A. Verify Package Compatibility

- **TensorFlow and NumPy Compatibility**: Ensure that the versions of TensorFlow and NumPy are compatible.
- **Avoid Unnecessary Packages**: If `scikit-learn` is not required for inference in your Flask app, remove it from `requirements.txt`.

B. Use Pre-built Wheels

To avoid compiling packages from source, use versions that have pre-built wheels for your Python version.

- **Specify Compatible Versions**: For Python 3.9, use package versions that are known to have pre-built wheels.

C. Alternative: Use a Non-slim Base Image

If you prefer not to install build tools, you can use the full Python image:

Alternative `Dockerfile` Base Image:

dockerfile
Copy code

```
FROM python:3.9
```

- **Pros**: Contains build tools by default.
- **Cons**: Larger image size.

Summary

- **Error Cause**: Incompatibility issues due to Python version, missing build tools, and package versions.
- **Solution**:
  - Use a stable Python version (e.g., 3.9).
  - Ensure all package versions in `requirements.txt` are compatible.
  - Install necessary build tools and Cython if needed.
  - Remove unnecessary packages to simplify the build.

Let Me Know If You Need Further Assistance

If you encounter any other issues or have questions about any of the steps, please don't hesitate to ask. I'm here to help you successfully complete Phase 2 of your project.

You said:

Ok I finally built the docker image. Before running docker run -p 5000:5000 mtbogush/model-api:v1 do i need to adjust any files?

Thought for 17 seconds

Congratulations on successfully building your Docker image! Before you run your container with `docker run -p 5000:5000 mtbogush/model-api:v1`, let's make sure everything is set up correctly to ensure your Flask app runs smoothly inside the container.

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Checklist Before Running the Docker Container

1. **Verify All Necessary Files Are Included**
    - **Model File (`lstm_model.h5`):**
        - Ensure that `lstm_model.h5` is in the same directory as your `Dockerfile` and your Flask app (`starter_notebook_phase_2.py`).
        - Since your `Dockerfile` includes `COPY . /app`, all files in the current directory, including your model file, are copied into the Docker image.
        - **No adjustments needed** if the model file is already in the correct location.
    - **Flask App File (`starter_notebook_phase_2.py`):**
        - Confirm that your Flask app file is named `starter_notebook_phase_2.py` as referenced in your `Dockerfile`.
        - If you've renamed your Flask app to `app.py`, update the `CMD` line in your `Dockerfile` accordingly.
    - **Requirements File (`requirements.txt`):**
        - Ensure that `requirements.txt` includes all necessary dependencies and is up-to-date.
2. **Double-Check Your `Dockerfile`**
    - **CMD Instruction:**
        - Your `Dockerfile` should have the following line:

          dockerfile
          Copy code
          ```dockerfile
          CMD ["python", "starter_notebook_phase_2.py"]
          ```
        - If your Flask app is named differently, adjust the filename in the `CMD` instruction.
    - **Base Image and Dependencies:**
        - Ensure your base image (`FROM python:3.9-slim`) is appropriate.
        - Confirm that any build tools and packages are correctly installed as per previous guidance.
3. **Environment Variables and Configurations**
    - If your application relies on any environment variables, ensure they are set within the Dockerfile or Kubernetes deployment files.
    - For a basic Flask app, this may not be necessary.

Testing the Docker Container Locally

Before deploying your application, it's essential to test it locally.

1. Run the Docker Container

bash
Copy code
```bash
docker run -p 5000:5000 mtbogush/model-api:v1
```

- This command runs your Docker container and maps port `5000` of your host machine to port `5000` of the container.
- If your Flask app uses a different port, adjust the command accordingly.

2. Verify the Container is Running

- In your terminal, you should see output from the Flask app indicating that it's running, similar to:

  csharp
  Copy code
  ```csharp
  * Serving Flask app 'starter_notebook_phase_2.py'
  * Running on all addresses (0.0.0.0)
  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
  ```

3. Send a Test Request to the API

- Use `curl` or a tool like Postman to send a POST request to your `/predict` endpoint.

Example `curl` Command:

bash
Copy code
```bash
curl -X POST -H "Content-Type: application/json" \
-d '{"input": [[0.1, 0.2, ..., 0.207], ..., [0.1, 0.2, ..., 0.207]]}' \
http://localhost:5000/predict
```

- Replace `...` with actual numerical values.
- Ensure your input data is shaped correctly:
    - For your model expecting input shape `(10, 207)`:

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

        - Provide 10 lists (time steps), each containing 207 numerical values (features).

Sample Input Data:

json

Copy code

```json
{
  "input": [
    [0.1, 0.2, ..., 0.207],  // First time step
    [0.1, 0.2, ..., 0.207],  // Second time step
    // ... 8 more time steps
  ]
}
```

4. Check the Response

- The API should return a JSON response with the prediction.

Example Response:

json

Copy code

```json
{
  "prediction": [[0.5]]
}
```

- If you receive an error, the response might look like:

  json

  Copy code

  ```json
  {
    "error": "Error message detailing what went wrong."
  }
  ```

5. Troubleshoot Any Issues

- **Common Issues and Solutions:**
    - **Error: Model File Not Found**
        - Ensure `lstm_model.h5` is present in the `/app` directory inside the container.
        - Verify the path in your Flask app's `load_trained_model` function is correct.
    - **Error: Incorrect Input Shape**
        - Confirm that your input data matches the expected shape `(1, 10, 207)`.
        - Adjust your input data accordingly.
    - **Dependency Errors**
        - Ensure all dependencies are correctly listed in `requirements.txt`.
        - Rebuild the Docker image if you make changes.
- **Check Container Logs:**
    - Look at the terminal output where the Docker container is running for any error messages.

Adjustments If Necessary

- **Update Flask App Filename:**
    - If your Flask app is named `app.py`, update the `CMD` instruction in your `Dockerfile`:

      dockerfile

      Copy code

      ```dockerfile
      CMD ["python", "app.py"]
      ```
- **Rebuild Docker Image After Changes:**
    - If you make any changes to your files, rebuild the Docker image:

      bash

      Copy code

      ```bash
      docker build -t mtbogush/model-api:v1 .
      ```

Next Steps

1. Push the Docker Image to DockerHub

- Once you've confirmed that your Docker container works locally, push the image to DockerHub:

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

bash
Copy code

```
docker login
docker push mtbogush/model-api:v1
```

- Ensure your `docker_image_link.txt` file contains the correct link to your DockerHub repository.

2. Deploy to Kubernetes

- Use your `deployment.yaml` and `service.yaml` files to deploy your application.

Apply Kubernetes Configurations:

bash
Copy code

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

- **Verify Deployment:**

  bash
  Copy code

  ```
  kubectl get pods
  kubectl get services
  ```

3. Test the Deployed Application

- Obtain the external IP address of your service (if using a LoadBalancer):

  bash
  Copy code

  ```
  kubectl get service traffic-prediction-service
  ```

- Send a test request to the deployed API endpoint using the external IP.

Example:

bash
Copy code

```
curl -X POST -H "Content-Type: application/json" \
-d '{"input": [[...]]}' \
http://EXTERNAL-IP-ADDRESS/predict
```

4. Update Documentation

- Ensure your `api_documentation.pdf` reflects any changes made.
- Include instructions on how to interact with the deployed API.

Final Checks

- **All Files Are Updated and Correct:**
  - `Dockerfile` matches your application files.
  - Necessary files are included in the build context.
- **Testing Completed Successfully:**
  - Both local and deployed versions of your application are functioning as expected.
- **Deliverables Are Prepared:**
  - `docker_image_link.txt` contains the correct DockerHub link.
  - `kubernetes_deployment_files.zip` includes your YAML files.
  - `api_documentation.pdf` is up-to-date.

Conclusion

No further adjustments should be necessary if:

- All your files are correctly set up and included.
- You've tested your Docker image locally and it works as expected.

You can proceed to run your Docker container:

bash
Copy code

```
docker run -p 5000:5000 mtbogush/model-api:v1
```

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

If you encounter any issues during testing or deployment, feel free to ask for assistance. Good luck with the rest of your project!

You said:

I get this error. Here are the relevant paths: C:\foai\lstm_model.h5 C:\foai\94879-starter-code-Team-Project\Dockerfile C:\foai\94879-starter-code-Team-Project\starter_notebook_phase_2.py Am i supposed to use a different path? ERROR: (OpAI_VM) C:\foai\94879-starter-code-Team-Project>docker run -p 5000:5000 mtbogush/model-api:v1 2024-10-08 05:21:11.097218: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`. 2024-10-08 05:21:11.097649: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used. 2024-10-08 05:21:11.099792: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used. 2024-10-08 05:21:11.105778: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered 2024-10-08 05:21:11.116237: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered 2024-10-08 05:21:11.119446: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered 2024-10-08 05:21:11.141564: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations. To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags. 2024-10-08 05:21:11.960565: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT Traceback (most recent call last): File "/app/starter_notebook_phase_2.py", line 12, in <module> model = load_trained_model() File "/app/starter_notebook_phase_2.py", line 8, in load_trained_model return load_model(model_path) File "/usr/local/lib/python3.9/site-packages/keras/src/saving/saving_api.py", line 196, in load_model return legacy_h5_format.load_model_from_hdf5(File "/usr/local/lib/python3.9/site-packages/keras/src/legacy/saving/legacy_h5_format.py", line 116, in load_model_from_hdf5 f = h5py.File(filepath, mode="r") File "/usr/local/lib/python3.9/site-packages/h5py/_hl/files.py", line 561, in __init__ fid = make_fid(name, mode, userblock_size, fapl, fcpl, swmr=swmr) File "/usr/local/lib/python3.9/site-packages/h5py/_hl/files.py", line 235, in make_fid fid = h5f.open(name, flags, fapl=fapl) File "h5py/_objects.pyx", line 54, in h5py._objects.with_phil.wrapper File "h5py/_objects.pyx", line 55, in h5py._objects.with_phil.wrapper File "h5py/h5f.pyx", line 102, in h5py.h5f.open FileNotFoundError: [Errno 2] Unable to synchronously open file (unable to open file: name = 'lstm_model.h5', errno = 2, error message = 'No such file or directory', flags = 0, o_flags = 0) (OpAI_VM) C:\foai\94879-starter-code-Team-Project>docker run -p 5000:5000 mtbogush/model-api:v1 2024-10-08 05:22:10.782855: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`. 2024-10-08 05:22:10.783549: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used. 2024-10-08 05:22:10.787077: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used. 2024-10-08 05:22:10.794996: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered 2024-10-08 05:22:10.814124: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered 2024-10-08 05:22:10.819520: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered 2024-10-08 05:22:10.828801: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations. To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags. 2024-10-08 05:22:11.594432: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT Traceback (most recent call last): File "/app/starter_notebook_phase_2.py", line 12, in <module> model = load_trained_model() File "/app/starter_notebook_phase_2.py", line 8, in load_trained_model return load_model(model_path) File "/usr/local/lib/python3.9/site-packages/keras/src/saving/saving_api.py", line 196, in load_model return legacy_h5_format.load_model_from_hdf5(File "/usr/local/lib/python3.9/site-packages/keras/src/legacy/saving/legacy_h5_format.py", line 116, in load_model_from_hdf5 f = h5py.File(filepath, mode="r") File "/usr/local/lib/python3.9/site-packages/h5py/_hl/files.py", line 561, in __init__ fid = make_fid(name, mode, userblock_size, fapl, fcpl, swmr=swmr) File "/usr/local/lib/python3.9/site-packages/h5py/_hl/files.py", line 235, in make_fid fid = h5f.open(name, flags, fapl=fapl) File "h5py/_objects.pyx", line 54, in h5py._objects.with_phil.wrapper File "h5py/_objects.pyx", line 55, in h5py._objects.with_phil.wrapper File "h5py/h5f.pyx", line 102, in h5py.h5f.open FileNotFoundError: [Errno 2] Unable to synchronously open file (unable to open file: name = 'lstm_model.h5', errno = 2, error message = 'No such file or directory', flags = 0, o_flags = 0) (OpAI_VM) C:\foai\94879-starter-code-Team-Project>

Thought for 16 seconds

I understand you're encountering a `FileNotFoundError` when running your Docker container, specifically indicating that the `lstm_model.h5` file is not found inside the container. Let's diagnose the issue and provide a solution.

Understanding the Error

The error message you're seeing is:

lua

Copy code

```lua
FileNotFoundError: [Errno 2] Unable to synchronously open file (unable to open file: name = 'lstm_model.h5', errno = 2, error message = '
```

This indicates that your Flask application inside the Docker container cannot find the `lstm_model.h5` file at the expected location.

Analyzing the File Paths

From your message, the relevant files are located at:

- **Model File (`lstm_model.h5`):**

  makefile

  Copy code

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
C:\foai\lstm_model.h5
```

- **Dockerfile and Flask App:**

  makefile
  Copy code
  ```
  C:\foai\94879-starter-code-Team-Project\Dockerfile
  C:\foai\94879-starter-code-Team-Project\starter_notebook_phase_2.py
  ```

This means that:

- The **model file** is in `C:\foai\`
- The **Dockerfile** and **Flask app** are in `C:\foai\94879-starter-code-Team-Project\`

When you run the `docker build` command, you're likely doing it from `C:\foai\94879-starter-code-Team-Project\`, which means that the build context is this directory.

Understanding Docker Build Context

- **Docker Build Context:** The directory you specify when running `docker build` (the `.` at the end of the command) defines the build context. Docker can only see files and directories within this context during the build process.

- **COPY Command in Dockerfile:** The command `COPY . /app` copies all files from the build context into the `/app` directory in the container.

Therefore, any files outside of the build context are not available during the Docker build process and won't be included in your image.

Solution

Option 1: Move the Model File into the Build Context

Steps:

1. **Copy or Move `lstm_model.h5` into the Project Directory**

   - Copy `lstm_model.h5` from `C:\foai\` to `C:\foai\94879-starter-code-Team-Project\`

     bash
     Copy code
     ```
     # From command prompt or terminal
     copy C:\foai\lstm_model.h5 C:\foai\94879-starter-code-Team-Project\
     ```

   - Alternatively, you can manually drag and drop the file using File Explorer.

2. **Verify the File Location**

   - Ensure that `lstm_model.h5` is now located at:

     makefile
     Copy code
     ```
     C:\foai\94879-starter-code-Team-Project\lstm_model.h5
     ```

3. **Rebuild the Docker Image**

   - Since the Docker image doesn't include the model file yet, you need to rebuild it:

     bash
     Copy code
     ```
     cd C:\foai\94879-starter-code-Team-Project\
     docker build -t mtbogush/model-api:v1 .
     ```

4. **Run the Docker Container**

   bash
   Copy code
   ```
   docker run -p 5000:5000 mtbogush/model-api:v1
   ```

5. **Test the Application**

   - Send a test request to ensure the application is working correctly.

Option 2: Adjust the Dockerfile to Include the Model File from Its Current Location

If you prefer to keep your model file in its current location, you can adjust your Dockerfile to copy it from the parent directory.

However, **this approach requires adjusting the build context and is generally not recommended** because Docker cannot access files outside the build context.

But if you must, here's how you could do it:

1. **Change the Build Context**

   - Run the Docker build command from `C:\foai\` instead, and specify the path to the Dockerfile.

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

    bash
    Copy code
    ```
    cd C:\foai\
    docker build -f 94879-starter-code-Team-Project\Dockerfile -t mtbogush/model-api:v1 .
    ```

2. **Adjust the `COPY` Commands in the Dockerfile**
    - Modify the `Dockerfile` to reflect the new paths:

      dockerfile
      Copy code
      ```
      # Copy the model file into the container
      COPY lstm_model.h5 /app/

      # Copy the application code into the container
      COPY 94879-starter-code-Team-Project/starter_notebook_phase_2.py /app/
      COPY 94879-starter-code-Team-Project/requirements.txt /app/
      ```

3. **Rebuild the Docker Image**
    - Run the adjusted `docker build` command as above.

4. **Run the Docker Container and Test**
    - Proceed as before.

Note: This method is more complex and can lead to confusion. I strongly recommend **Option 1** for simplicity.

Recommendation

Use Option 1: Move or copy `lstm_model.h5` into your project directory (`C:\foai\94879-starter-code-Team-Project\`), so it's included in the Docker build context.

Detailed Steps

1. Move the Model File

- **Copy `lstm_model.h5` to Your Project Directory**

  bash
  Copy code
  ```
  copy C:\foai\lstm_model.h5 C:\foai\94879-starter-code-Team-Project\
  ```

2. Verify File Structure

Your project directory (`C:\foai\94879-starter-code-Team-Project\`) should now contain:

- `Dockerfile`
- `starter_notebook_phase_2.py`
- `lstm_model.h5`
- `requirements.txt`
- Any other necessary files

3. Rebuild the Docker Image

bash
Copy code
```
cd C:\foai\94879-starter-code-Team-Project\
docker build -t mtbogush/model-api:v1 .
```

4. Run the Docker Container

bash
Copy code
```
docker run -p 5000:5000 mtbogush/model-api:v1
```

5. Test the Application

- Use `curl` or Postman to send a request to `http://localhost:5000/predict` with appropriate input data.

Why This Works

- **Including the Model in the Build Context:**
    - By moving `lstm_model.h5` into the project directory, it's now part of the build context when you run `docker build`.
- **COPY Command in Dockerfile:**

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

  - The `COPY . /app` command copies all files in the current directory (the build context) into `/app` inside the container, including your model file.
- **Application Can Find the Model File:**
  - Inside the container, your Flask app expects `lstm_model.h5` to be in the `/app` directory (since the working directory is `/app`), and now it is.

Additional Tips

- **Check the Working Directory:**
  - In your Dockerfile, you have:

    dockerfile
    Copy code
    ```
    WORKDIR /app
    ```
  - This means that when your Flask app runs, it's operating in the `/app` directory inside the container.
- **Relative Paths in Code:**
  - In your Flask app, the `load_trained_model` function uses a relative path:

    python
    Copy code
    ```
    def load_trained_model(model_path='lstm_model.h5'):
        return load_model(model_path)
    ```
  - This will look for `lstm_model.h5` in the current working directory (`/app` inside the container).
- **Ensuring All Necessary Files Are Copied:**
  - The `COPY . /app` command in the Dockerfile ensures that all files in your project directory are included in the container.

Verifying the Fix

- **Successful Start-Up:**
  - After rebuilding the image and running the container, your Flask app should start without the `FileNotFoundError`.
- **Application Logs:**
  - Check the logs outputted in your terminal when the container runs. You should see the standard Flask start-up messages without errors.
- **Testing the API:**
  - Send a test request to confirm that the model is loaded correctly and the API returns predictions.

Conclusion

By ensuring that the `lstm_model.h5` file is included in your Docker build context and copied into the container, you resolve the `FileNotFoundError` and allow your Flask application to load the model successfully inside the Docker container.

Let me know if you encounter any other issues or need further assistance!

You said:

what is going on? (OpAI_VM) C:\foai\94879-starter-code-Team-Project>docker run -p 5000:5000 mtbogush/model-api:v1 2024-10-08 05:26:14.891707: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`. 2024-10-08 05:26:14.892734: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used. 2024-10-08 05:26:14.896222: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used. 2024-10-08 05:26:14.904967: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:485] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered 2024-10-08 05:26:14.920061: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:8454] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered 2024-10-08 05:26:14.924079: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1452] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered 2024-10-08 05:26:14.936036: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations. To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags. 2024-10-08 05:26:15.926547: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model. * Serving Flask app 'starter_notebook_phase_2' * Debug mode: off INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead. * Running on all addresses (0.0.0.0) * Running on http://127.0.0.1:5000 * Running on http://172.17.0.2:5000 INFO:werkzeug:Press CTRL+C to quit PS C:\foai\94879-starter-code-Team-Project> docker login Authenticating with existing credentials... Login Succeeded PS C:\foai\94879-starter-code-Team-Project> docker push mtbogush/model-api:v1 The push refers to repository [docker.io/mtbogush/model-api] 58010565938e:

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
Pushing [==================================================>] 60.59MB/60.59MB 7ca8d966a6b3: Retrying in 20 seconds 233a6b1c2403: Pushed a3db373ea63e: Pushed 3eb641edd5f4: Retrying in 20 seconds a62be9b8a271: Pushed 9e599118e168: Mounted from library/python e228adf1886f: Mounted from library/python fb5ccd0db472: Mounted from library/python 8d853c8add5d: Mounted from library/python use of closed network connection PS C:\foai\94879-starter-code-Team-Project> kubectl apply -f deployment.yaml error: the path "deployment.yaml" does not exist PS C:\foai\94879-starter-code-Team-Project> kubectl get pods E1008 01:31:00.569240 26024 memcache.go:265] couldn't get current server API group list: Get "https://127.0.0.1:60169/api?timeout=32s": dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively refused it. E1008 01:31:00.570311 26024 memcache.go:265] couldn't get current server API group list: Get "https://127.0.0.1:60169/api?timeout=32s": dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively refused it. E1008 01:31:00.571824 26024 memcache.go:265] couldn't get current server API group list: Get "https://127.0.0.1:60169/api?timeout=32s": dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively refused it. E1008 01:31:00.572392 26024 memcache.go:265] couldn't get current server API group list: Get "https://127.0.0.1:60169/api?timeout=32s": dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively refused it. E1008 01:31:00.578246 26024 memcache.go:265] couldn't get current server API group list: Get "https://127.0.0.1:60169/api?timeout=32s": dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively refused it. Unable to connect to the server: dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively refused it. PS C:\foai\94879-starter-code-Team-Project> kubectl get service traffic-prediction-service >> E1008 01:31:20.144867 9008 memcache.go:265] couldn't get current server API group list: Get "https://127.0.0.1:60169/api?timeout=32s": dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively refused it. E1008 01:31:20.145865 9008 memcache.go:265] couldn't get current server API group list: Get "https://127.0.0.1:60169/api?timeout=32s": dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively refused it. E1008 01:31:20.159152 9008 memcache.go:265] couldn't get current server API group list: Get "https://127.0.0.1:60169/api?timeout=32s": dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively refused it. E1008 01:31:20.160152 9008 memcache.go:265] couldn't get current server API group list: Get "https://127.0.0.1:60169/api?timeout=32s": dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively refused it. E1008 01:31:20.175016 9008 memcache.go:265] couldn't get current server API group list: Get "https://127.0.0.1:60169/api?timeout=32s": dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively refused it. Unable to connect to the server: dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively refused it. PS C:\foai\94879-starter-code-Team-Project>
```

Thought for 15 seconds

I'm sorry to hear you're encountering these issues. Let's go through each error step by step to understand what's happening and how to resolve it.

Summary of the Issues

1. **Docker Run Output:**
    - Your Flask app appears to be running inside the Docker container.
    - The application logs indicate that it's running on `http://127.0.0.1:5000` and `http://172.17.0.2:5000`.
2. **Docker Push Error:**
    - The Docker push operation is failing with messages like `Retrying in 20 seconds` and `use of closed network connection`.
3. **Kubernetes Deployment Issues:**
    - When running `kubectl apply -f deployment.yaml`, you get an error: `the path "deployment.yaml" does not exist`.
    - When running `kubectl get pods`, you receive connection errors: `Unable to connect to the server: dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively refused it.`

---

1. Docker Run Output

Your Docker container is running, and the Flask app is starting without errors related to missing files. This suggests that the earlier issue with the `lstm_model.h5` file has been resolved.

Application Output:

vbnet

Copy code

```vbnet
 * Serving Flask app 'starter_notebook_phase_2'
 * Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000
INFO:werkzeug:Press CTRL+C to quit
```

Action Items:

- **Test the API Locally:**
    - While the container is running, send a test request to `http://localhost:5000/predict` to verify that the API is functioning correctly.
    - Use `curl` or Postman with appropriate input data.

---

2. Docker Push Error

Error Messages:

makefile

Copy code

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

```
7ca8d966a6b3: Retrying in 20 seconds
3eb641edd5f4: Retrying in 20 seconds
use of closed network connection
```

This indicates that Docker is having trouble pushing some layers to DockerHub, and eventually the connection is closed.

Possible Causes:

- **Network Connectivity Issues:**
  - Unstable internet connection causing interruptions during the push.
- **DockerHub Rate Limits:**
  - Exceeding the number of allowed pushes in a certain time frame.
- **Large Image Size:**
  - If your Docker image is large (e.g., due to including the model file), pushing can take longer and be more susceptible to timeouts.
- **Authentication Issues:**
  - Session expired or credentials are invalid.

Solutions:

1. **Check Your Internet Connection:**
   - Ensure you have a stable internet connection.
2. **Log Out and Log Back In to DockerHub:**

   bash
   Copy code
   ```
   docker logout
   docker login
   ```
   - Re-enter your DockerHub username and password.
3. **Retry the Push Operation:**

   bash
   Copy code
   ```
   docker push mtbogush/model-api:v1
   ```
4. **Check DockerHub Status:**
   - Visit DockerHub Status to see if there are any ongoing issues.
5. **Reduce Image Size (Optional):**
   - If your image is very large, consider optimizing it:
     - Use a smaller base image.
     - Clean up unnecessary files.
     - Use multi-stage builds.
6. **Use a Different Tag:**
   - Sometimes re-tagging the image can help:

     bash
     Copy code
     ```
     docker tag mtbogush/model-api:v1 mtbogush/model-api:v2
     docker push mtbogush/model-api:v2
     ```

Additional Tips:

- **Check for Firewall or Antivirus Interference:**
  - Ensure that your firewall or antivirus software isn't blocking Docker's network connections.

---

3. Kubernetes Deployment Issues

a. Error: The path "deployment.yaml" does not exist

Cause:

- The `deployment.yaml` file is not in the current directory (`C:\foai\94879-starter-code-Team-Project\`).

Solution:

1. **Verify the Location of `deployment.yaml` and `service.yaml`:**
   - Ensure that these files are present in the current directory.

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

bash
Copy code
```
dir C:\foai\94879-starter-code-Team-Project\
```

2. **If They Are in a Different Directory:**
   - Navigate to the directory where the files are located.

     bash
     Copy code
     ```
     cd C:\path\to\directory\containing\yaml\files
     ```
   - Or provide the full path to the file when running the command:

     bash
     Copy code
     ```
     kubectl apply -f C:\path\to\deployment.yaml
     ```
3. **Ensure Correct File Names:**
   - Check for typos in the file names.

b. Error: Unable to Connect to the Server

Error Messages:

css
Copy code
```
Unable to connect to the server: dial tcp 127.0.0.1:60169: connectex: No connection could be made because the target machine actively re
```

Cause:

- **Kubernetes Cluster Is Not Running:**
  - The `kubectl` command cannot connect to the Kubernetes API server because it's not running.
- **Misconfigured `kubectl` Context:**
  - The `kubectl` command may be pointing to a cluster that doesn't exist or is not accessible.

Solutions:

1. **Ensure Kubernetes Is Installed and Running:**
   - **If Using Docker Desktop:**
     - Open Docker Desktop.
     - Go to Settings > Kubernetes.
     - Ensure that "Enable Kubernetes" is checked.
     - If not, check it and apply changes.
     - Wait for Kubernetes to start.
   - **If Using Minikube:**
     - Start Minikube:

       bash
       Copy code
       ```
       minikube start
       ```
   - **If Using Other Kubernetes Solutions:**
     - Ensure your Kubernetes cluster is up and running.
2. **Check `kubectl` Configuration:**
   - Verify the current context:

     bash
     Copy code
     ```
     kubectl config current-context
     ```
   - List all contexts:

     bash
     Copy code
     ```
     kubectl config get-contexts
     ```
   - Ensure the context is set to the correct cluster.
3. **Check Kubernetes Cluster Status:**

   bash

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Copy code

```
kubectl cluster-info
```

- This should provide information about the cluster if it's running.

4. **Restart Docker Desktop (If Applicable):**
   - Sometimes, restarting Docker Desktop can resolve connectivity issues.
5. **Check for Firewall or Antivirus Interference:**
   - Ensure that your firewall or antivirus software isn't blocking Kubernetes components.

Additional Steps:

- **Update Kubernetes Configuration:**
  - If your Kubernetes configuration (`~/.kube/config`) is corrupted or misconfigured, you might need to reset it.
- **Reinstall Kubernetes Components:**
  - As a last resort, you might consider reinstalling Kubernetes or Docker Desktop.

---

4. Next Steps

A. Fix the Docker Push Issue

1. **Retry the Push After Checking Connection and Credentials**
   - Ensure you're logged in:

     bash
     Copy code

     ```
     docker logout
     docker login
     ```

   - Retry the push:

     bash
     Copy code

     ```
     docker push mtbogush/model-api:v1
     ```

2. **Monitor the Push Progress**
   - If it stalls or fails again, consider pushing during a time when network traffic is lower.

B. Ensure Kubernetes Cluster Is Running

1. **Start the Kubernetes Cluster**
   - **Docker Desktop:**
     - Start Docker Desktop.
     - Enable Kubernetes in the settings if not already enabled.
     - Wait for Kubernetes to start (this may take a few minutes).
   - **Minikube:**

     bash
     Copy code

     ```
     minikube start
     ```

2. **Verify Kubernetes Is Running**
   - Check cluster info:

     bash
     Copy code

     ```
     kubectl cluster-info
     ```

   - Check nodes:

     bash
     Copy code

     ```
     kubectl get nodes
     ```

C. Apply Kubernetes Deployment

1. **Navigate to the Directory Containing the YAML Files**

   bash

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

   Copy code

   ```
   cd C:\foai\94879-starter-code-Team-Project\
   ```

   - Ensure `deployment.yaml` and `service.yaml` are present.

2. **Apply the Deployment and Service**

   bash

   Copy code

   ```
   kubectl apply -f deployment.yaml
   kubectl apply -f service.yaml
   ```

3. **Verify the Deployment**

   bash

   Copy code

   ```
   kubectl get deployments
   kubectl get pods
   kubectl get services
   ```

D. Test the Deployed Application

1. **Get the Service Details**

   bash

   Copy code

   ```
   kubectl get service traffic-prediction-service
   ```

   - Note the external IP address if available.

2. **Send a Test Request**

   - Use the external IP to send a request to your API endpoint.

---

5. Additional Recommendations

A. Use Proper Tools for Production

- The warning in your Flask app logs:

  vbnet

  Copy code

  ```
  WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
  ```

- **Recommendation:**
  - For production deployments, consider using a production-ready server like **Gunicorn** or **uWSGI**.

- **Example Adjustments:**
  - **Install Gunicorn:**

    Add to your `requirements.txt`:

    Copy code

    ```
    gunicorn
    ```

  - **Update Your Dockerfile CMD:**

    dockerfile

    Copy code

    ```
    CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:5000", "starter_notebook_phase_2:app"]
    ```

    - This command tells Gunicorn to start 4 worker processes and bind to `0.0.0.0:5000`.

B. Monitor Resource Usage

- Ensure your Kubernetes cluster has sufficient resources to run your application.
- Check pod statuses for any crashes or restarts.

C. Check for Rate Limits

- Be aware of DockerHub's rate limits, especially if you're on a free plan.
- If necessary, consider using a private registry or upgrading your DockerHub plan.

---

6. Conclusion

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF

Summary of Actions:

- **Docker Push Issue:**
    - Retry the push after ensuring stable network connectivity and valid authentication.
    - Check for any firewall or antivirus interference.
- **Kubernetes Deployment Issues:**
    - Ensure Kubernetes is installed and running.
    - Verify that your `deployment.yaml` and `service.yaml` files are in the correct directory.
    - Apply the deployment and service, and test the application.

Let me know if you need further assistance or if any of these steps are unclear. I'm here to help you get your application running smoothly!

ChatGPT can make mistakes. Check important info.

?

Explore our developer-friendly HTML to PDF API Printed using PDFCrowd HTML to PDF