

# Holistic Traffic Prediction for Smart Cities: A Full-Cycle Approach

Peter Muller, Ariana Rocha, Ayush Tripathi,  
Katie Burgess, and Goutam Mukku

# Introduction

This comprehensive project focuses on using the METR-LA, focusing on the operationalization of machine learning models in a real-world context. This assignment involved experimentation, deployment, and monitoring phases, leveraging advanced tools such as Kubeflow, Kubernetes, Docker, and Evidently.



# Dataset & Preprocessing Steps

- METR-LA dataset:
  - Traffic speed data collected from 207 loop detectors in LA every 5 minutes
  - Time series data
- We cleaned and normalized data to prepare for the models
  - Sliding window applied, capture temporal relationships

## Data Cleaning and Normalization

```
df_scaled = pd.read_csv('/Users/ayushtripathi/Downloads/metr_la_data_with_headers.csv')

# Step 1: Data Cleaning
# Manually forward-fill columns where there are zero values to avoid recursion errors
df_cleaned_ffill = df_scaled.copy()

# Forward-fill or backward-fill any NaN values in the normalized dataset
df_scaled.fillna(method='ffill', inplace=True) # Forward fill NaN values
df_scaled.fillna(method='bfill', inplace=True) # Backward fill in case forward fill doesn't cover all NaNs

# Step 2: Data Normalization
scaler = MinMaxScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_cleaned_ffill), columns=df.columns)

print(df_scaled.head())
```

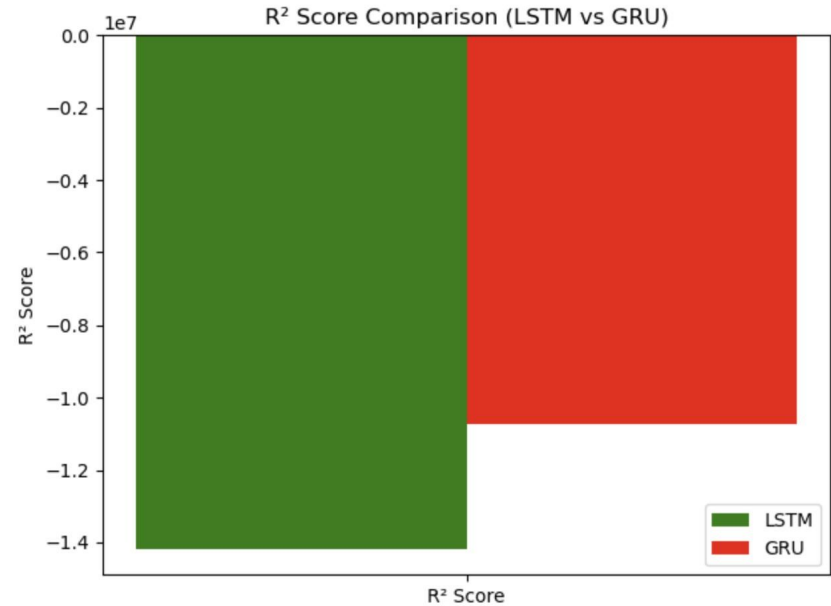
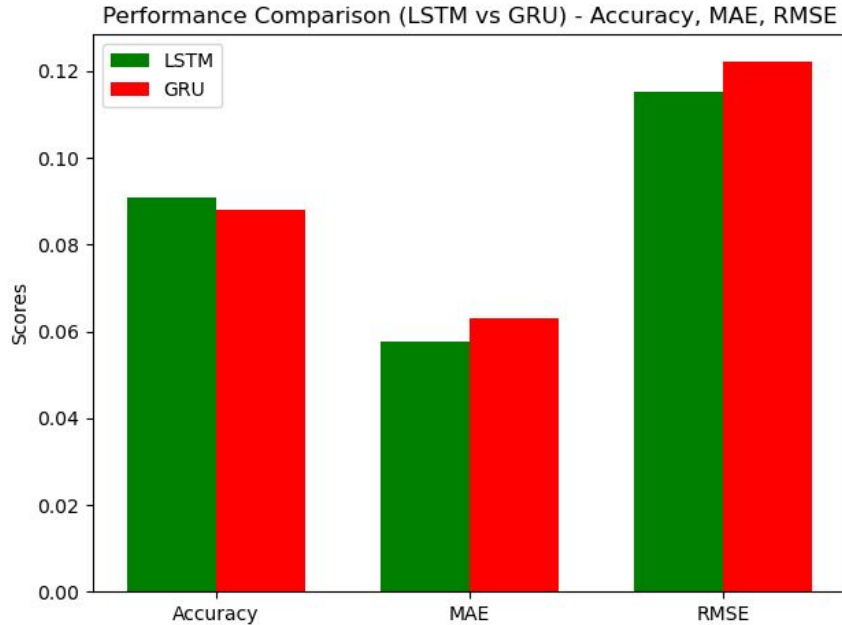
# Model Selection/Building/Evaluation

- We implemented LSTM and GRU models, built using TensorFlow
  - LSTM: Two LSTM layers with 64 units each, followed by dropout layers to prevent overfitting and a dense layer for output.
  - GRU: Two GRU layers with 64 units each, dropout layers, and a dense output layer.
- Models trained on 80% of data, 20% saved for testing
- Training conducted using the Adam optimizer with binary cross-entropy loss function and Sigmoid as activation function.

Model	MAE	RMSE	R2
LSTM	0.050	0.118	-1.4
GRU	0.056	0.121	-1.1

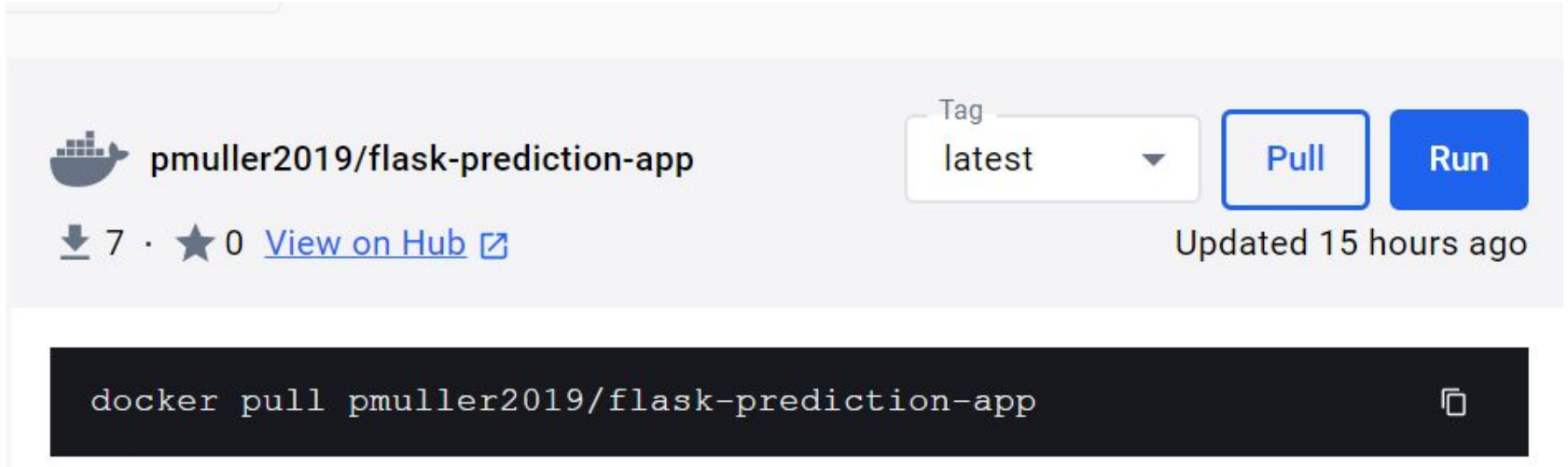
# LSTM & GRU Model comparison (Accuracy, MAE, RMSE, R<sup>2</sup>):

## Model Performance



Overall, the LSTM Model performed better than GRU

# Docker Usage



The screenshot shows the Docker Hub interface for the image `pmuller2019/flask-prediction-app`. On the left, there is a Docker logo, the image name, and statistics: 7 downloads and 0 stars, with a link to "View on Hub". On the right, there is a "Tag" dropdown menu set to "latest", a "Pull" button, and a "Run" button. Below these, it says "Updated 15 hours ago". At the bottom, a dark terminal window displays the command `docker pull pmuller2019/flask-prediction-app` with a copy icon on the right.

pmuller2019/flask-prediction-app

7 · ★ 0 [View on Hub](#)

Tag latest Pull Run

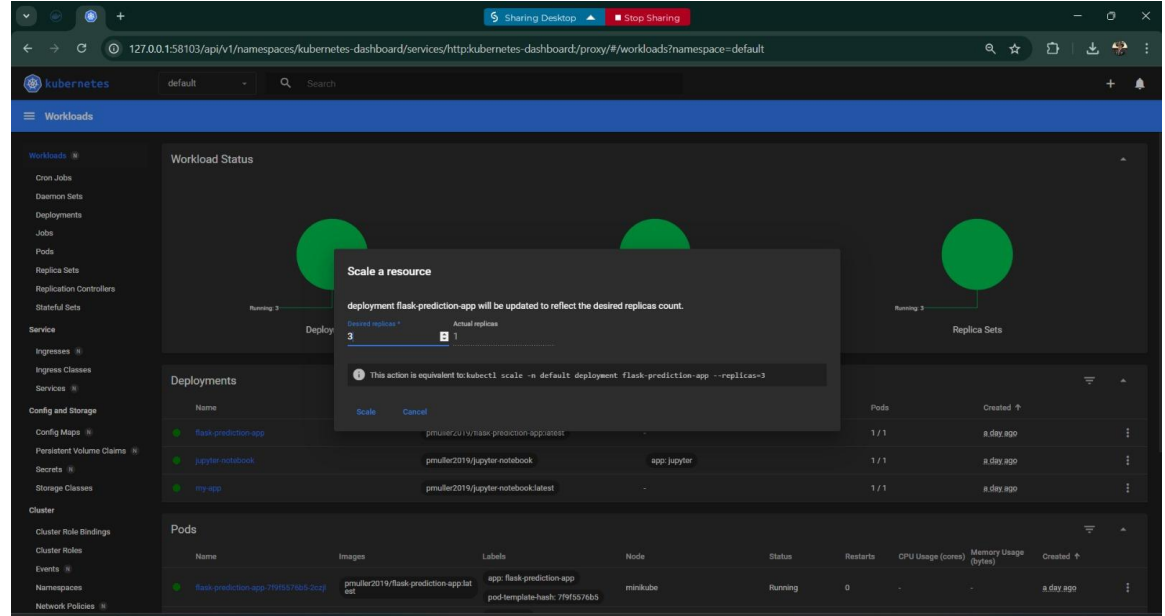
Updated 15 hours ago

```
docker pull pmuller2019/flask-prediction-app
```

- We host this image to run our prediction model both on Kube Flow and for our API
- Predictions are run repeatedly every 10 seconds to simulate live data

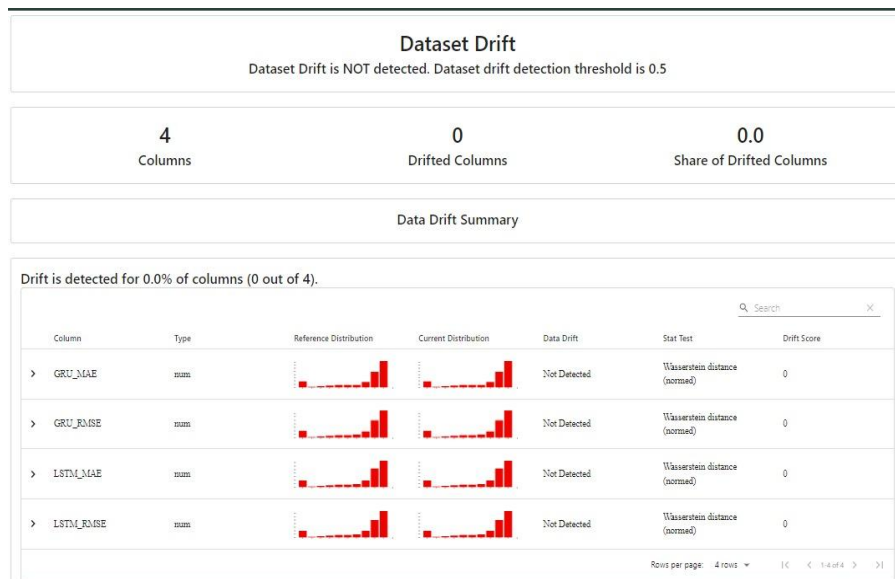
# Kubernetes Deployment: Model Delivery

- Deployed via yaml file
- Successfully deployed and able to see log files within display
- Originally had some issues due to Windows system
- Useful for model scalability



# RESTful API endpoint & Evidently: Model Monitoring

- You can interact with the model through a RESTful API via. Localhost
- Runs tests to correctly display if there is any drift in the model
  - Allows you to then update your model, features, and/or data to account for the changes to current inputted data





# Takeaways & Challenges

- Most of this process was new to all of us:
  - Kubeflow & Deploying Docker Images
  - Creating a RESTful API (Especially for ML Models)
  - Evidently
- We ran into some compatibility issues with Windows
- Splitting up our notebook into parts for the different project aspects took some trial and error

Q&A