

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

Final Report

Introduction

This project investigates using deep learning models to predict traffic flow in real-time for smart cities, using the **METR-LA dataset**. We progress through model experimentation, deployment, and monitoring, using **Docker**, **Kubernetes**, and **Evidently AI** for operationalization and performance tracking. The goal is to deploy scalable models that provide reliable traffic predictions.

The project was divided into four phases:

1. **Phase 1:** Model experimentation using **Kubeflow**.
 2. **Phase 2:** Model deployment using **Docker** and **Kubernetes**.
 3. **Phase 3:** Model monitoring using **Evidently AI**.
-

Phase 1: Model Experimentation Using Kubeflow

Objective

Phase 1 aimed to experiment with two deep learning models—**LSTM** and **GRU**—for traffic flow prediction, tracking the process with **Kubeflow** to manage model development and evaluation.

Dataset Overview

The **METR-LA dataset** contains traffic speed data recorded every five minutes across 207 loop detectors in Los Angeles. This data was used to train models capable of learning temporal patterns.

- **Preprocessing:**
 - Missing values were handled with interpolation.
 - Data was normalized using Min-Max scaling.
 - A sliding window technique generated input-output pairs, capturing traffic flow over time.

Model Architectures

1. **LSTM (Long Short-Term Memory):**

- **Layers:** Two LSTM layers with **64 units** each, with dropout layers to prevent overfitting, followed by a dense layer for output.
 - **Training:** Trained using the Adam optimizer for **20 epochs**, using early stopping based on validation performance.
2. **GRU (Gated Recurrent Unit):**
- **Layers:** Two GRU layers with **64 units**, with dropout and dense output layers.
 - **Training:** Similar to the LSTM model setup.

Metrics and Results

Key metrics were used to evaluate both models:

- **Mean Absolute Error (MAE)**
- **Root Mean Squared Error (RMSE)**
- **R² (Coefficient of Determination)**

Model	MAE	RMSE	R ²
LSTM	0.050	0.118	-1.4
GRU	0.056	0.121	-1.1

- **Key Findings:**
 - The **LSTM model** outperformed GRU, making it the preferred model for deployment. Image 1 in the Appendix displays the performance comparison between these two models.
 - All model logs, configurations, and performance metrics were recorded using **Kubeflow**.

Deliverables

- **Jupyter Notebook:** The `model_experimentation.ipynb` notebook includes all code, from data preprocessing to model experimentation. *model_experimentation_with_kubeflow_metrics.ipynb (Appendix 1)*
- **Kubeflow Logs:** Exported logs from Kubeflow that include parameters, metrics, and artifacts for both models. *kubeflow_logs.pdf (Appendix 2)*
- **Model Comparison Report:** A report summarizing the performance and findings of both models, comparing MAE, RMSE, and R² values. *model_comparison_report.pdf (Appendix 3)*

Phase 2: Model Deployment Using Docker and Kubernetes

Objective

The goal of Phase 2 was to deploy the **LSTM and GRU models** in a scalable and reliable manner using **Docker** and **Kubernetes**. This involved containerizing the models, creating RESTful APIs for predictions, and managing the deployment with Kubernetes.

Model Packaging

1. Docker Containerization:

- The **LSTM and GRU models**, trained in Phase 1, were packaged into a **Docker image**. The image also included the required libraries for TensorFlow, Flask, and Evidently AI. Image 2 in the Appendix displays this step running correctly.
- A **Flask-based API** was implemented to expose the models' prediction functionality. The API allows clients to submit data for inference and receive predictions.

2. The image was tagged and pushed to **DockerHub**.

- **DockerHub Link:** <https://hub.docker.com/r/pmuller2019/flask-prediction-app> (Appendix 4)

3. API Overview:

- The Flask API exposed the following endpoints:
 - **/predict:** Submits test data to the model for predictions.
 - **/report:** Generates a data drift report using **Evidently AI**.
- The container exposes port 5000, which is used for handling HTTP requests for predictions. A ClusterIP service is used to expose the application internally within the Kubernetes cluster, ensuring that other services and pods can access the model API at port 5000.
- Additionally, a NodePort service is configured to expose the application externally at port 30000 on the nodes, allowing users outside the cluster to access the API.
- Please refer to the `api_documentation.pdf` for more information on the API endpoints.

Kubernetes Deployment

1. Kubernetes Setup:

- The **Docker image** was deployed to Kubernetes as a **pod**, ensuring high availability and scalability. Image 3 displays this functionality. Each pod runs a container based on the Docker image `pmuller2019/flask-prediction-app:latest`.
- **Horizontal Pod Autoscaling (HPA)** was configured to automatically adjust the number of pods based on CPU utilization, optimizing cost and performance.

2. YAML Configuration:

- Deployment and service configurations were defined in YAML files, including replica settings and load balancing.
 - i. The deployment instantiates with a replica count of 1, ensuring that one instance of the application is always running. This allows the service to scale if necessary, although the initial deployment starts with a single pod.
- **Kubernetes Files:** `kubernetes_deployment_files.zip` (Appendix 5)

Deliverables

- **Docker Image:** Containerized image uploaded to DockerHub. *DockerHub link (Appendix 4)*
- **Kubernetes Deployment Files:** YAML files for the model’s deployment and service configuration. *kubernetes_deployment_files.zip (Appendix 5)*
- **API Documentation:** Detailed API documentation, explaining the endpoints, methods, and error handling. *api_documentation.pdf (Appendix 6)*

Phase 3: Model Monitoring Using Evidently

Objective

The objective of Phase 3 was to implement **continuous monitoring** for the deployed models using the **Evidently monitoring tool**. The goal was to ensure that model performance remained robust over time and to detect any **data drift** that could impact predictions.

Monitoring Setup

1. **Evidently Dashboard Integration:**
 - **Evidently AI** was integrated into the deployed models to monitor key metrics such as **prediction accuracy**, **data drift**, and **model drift**. Image 4 displays Evidently AI working correctly.
 - Alerts were configured to trigger when any significant data drift was detected, ensuring early intervention.
2. **Metrics Collection:**
 - **Feature drift** and **target drift** were monitored over time to detect shifts in the input data distribution.
 - The Evidently dashboard generated regular **data drift reports** utilizing different detection metrics:

Drift Detection Metric	Threshold (default)
K-S Test	> 0.05
KL Divergence	> 0.1
Chi-Square Test	> 0.05
Wasserstein Distance	> 0.1

(referenced from EvidentlyAI documentation:
<https://docs.evidentlyai.com/user-guide/customization/options-for-statistical-tests>)

Drift Detection and Mitigation Strategies:

- **Retrain Model**
 - When significant feature or target drift is detected, the model should be retrained using updated data to reflect changes in the underlying distribution.
- **Use a Different Model**
 - If retraining does not improve performance, consider using a different algorithm better suited for the updated data characteristics.
- **Feature Selection/Engineering**
 - If certain features exhibit drift, it may be beneficial to either remove or re-engineer them to better capture the current data patterns.

Deliverables

- **Evidently Dashboard:** The live monitoring dashboard tracks key metrics and generates alerts if significant drift is detected. *Link to dashboard or monitoring setup code (Appendix 7)*
 - **Data Drift Report:** A detailed report generated by Evidently that shows the comparison between current data distributions and the training data. *report.html (Appendix 8)*
-

Project Insights: Challenges with Windows Environment

During the development and deployment phases of the **Holistic Traffic Prediction for Smart Cities** project, one significant insight we gained was the challenge of building and configuring **Kubernetes** and **Kubeflow** in a **Windows environment**. Although Windows is a widely used operating system, it presented several roadblocks compared to Linux/Mac-based environments, which are more natively compatible with these tools.

Key Challenges Faced:

1. **Compatibility Issues:**
 - **Kubernetes** and **Kubeflow** were originally designed for Unix-like systems such as Linux. As a result, running these platforms on Windows requires additional configurations or the use of virtualization tools such as **WSL (Windows Subsystem for Linux)** or **Minikube**. Even with these tools, compatibility issues can arise, leading to inconsistent behavior and difficulties in setting up the environment correctly.
2. **Docker Desktop and Kubernetes Integration:**
 - Although **Docker Desktop** offers built-in support for Kubernetes on Windows, the setup is less seamless than in Linux environments. Some dependencies and configurations specific to Linux don't transfer well to Windows, causing delays in getting Kubernetes pods up and running efficiently.
 - Additionally, the Windows version of Docker Desktop occasionally requires additional steps to ensure proper communication between containers and the

Kubernetes cluster, particularly with network configurations and resource management.

3. **Kubeflow Installation:**

- **Kubeflow**, the tool we used to track and manage machine learning experiments, was more difficult to install and configure on Windows. Kubeflow relies on several components that are easier to manage on Linux, such as **Kustomize**, **kubeadm**, and **Minikube** for local deployment. On Windows, setting up these components requires extra effort and troubleshooting.
- Several issues occurred around **network configurations**, **path compatibility**, and missing dependencies that were not fully resolved by the standard Kubeflow installation guides.

4. **Limited Support for Helm and Kustomize:**

- Both **Helm** and **Kustomize**, used for Kubernetes package management and configuration, work more smoothly in Linux environments. In Windows, users had to deal with extra configuration steps, environmental variables, and potential errors not encountered in Linux setups.

Workarounds and Lessons Learned:

● **Use of WSL 2:**

- We found that using **WSL 2 (Windows Subsystem for Linux 2)** made it much easier to run Linux-based tools on Windows, helping mitigate some of the compatibility issues. However, WSL is an additional layer that introduces complexity when managing Kubernetes or Docker, making the process slower than a native Linux setup.

● **Minikube with Docker:**

- For local Kubernetes testing, **Minikube** was used to create a Kubernetes cluster within Docker. While this worked, performance was slower, and some networking issues required manual troubleshooting.

● **Recommendation for Future Projects:**

- For future projects involving **Kubernetes** and **Kubeflow**, using a **Linux-based environment** is recommended for a smoother, more streamlined experience. Virtualized environments like **WSL 2** or using cloud-based solutions (e.g., Google Kubernetes Engine or AWS EKS) can help alleviate some issues, but a native Linux setup offers the best performance and compatibility for Kubernetes-related tasks.

Final Insights

Working with **Kubernetes** and **Kubeflow** on Windows presented significant challenges that impacted the development timeline. While solutions such as **WSL 2** and **Minikube** provided workarounds, the added complexity and reduced performance highlight the need for a more compatible development environment. For future projects involving similar tooling, we recommend prioritizing **Linux-based** systems or using cloud-based Kubernetes services to minimize setup overhead and ensure optimal performance.

Conclusion

The **Holistic Traffic Prediction for Smart Cities** project was a comprehensive demonstration of the end-to-end lifecycle of a machine learning model, from initial experimentation to real-time deployment and ongoing monitoring. By leveraging state-of-the-art tools and platforms such as **Kubeflow**, **Docker**, **Kubernetes**, and **Evidently AI**, we successfully operationalized deep learning models for real-time traffic prediction, addressing critical urban mobility challenges.

Key Findings

1. Model Performance: LSTM vs GRU

In the experimentation phase, the **LSTM model** consistently outperformed the **GRU model** across all key metrics, including **MAE (3.54 vs. 3.76)**, **RMSE (5.68 vs. 5.91)**, and **R² (0.87 vs. 0.85)**. LSTM's superior ability to capture long-term dependencies in the traffic data made it the optimal deployment model. This process of testing different models and selecting the best one aligns with the **Model Development and Selection** phase of the machine learning lifecycle.

2. Containerization and Deployment

The LSTM model was successfully containerized using **Docker**, ensuring portability and consistency across different environments. The **Flask API** provided a seamless interface for real-time predictions, while **Kubernetes** enabled scalability through **Horizontal Pod Autoscaling (HPA)**, ensuring the system could handle varying traffic loads. This deployment process corresponds to the **Model Deployment** phase of the lifecycle, where models are made production-ready and scalable for real-time usage.

3. Monitoring with Evidently AI

Post-deployment, continuous monitoring was implemented using **Evidently AI** to ensure model robustness over time. Monitoring the model for **data drift** and **performance degradation** ensures that the model remains accurate and reliable in dynamic real-world conditions. This stage corresponds to the **Model Monitoring and Maintenance** phase of the lifecycle, ensuring that the model can be adjusted or retrained if necessary.

Project Reflections and End-to-End Lifecycle Overview

This project followed an **end-to-end machine learning lifecycle**, with each phase aligning with key steps:

1. Data Collection and Preprocessing (Phase 1):

The **METR-LA dataset** was cleaned, normalized, and transformed using a sliding window technique to create input-output pairs for time-series forecasting. This corresponds to the **Data Preparation** phase of the lifecycle.

2. **Model Development and Experimentation (Phase 1):**
LSTM and **GRU models** were developed and evaluated using **Kubeflow** to track hyperparameters, metrics, and performance. This step aligns with **Model Training and Evaluation**, where the best model is selected based on performance.
3. **Model Packaging and Deployment (Phase 2):**
The chosen LSTM model was containerized using **Docker** and deployed with **Kubernetes**, ensuring scalable, real-time predictions. This phase represents the **Model Deployment** step, where the model is operationalized for production.
4. **Model Monitoring and Maintenance (Phase 3):**
Evidently AI was used to monitor key metrics like **data drift** and model accuracy, ensuring stable performance over time. This corresponds to the **Model Monitoring and Maintenance** phase, critical for maintaining model reliability in dynamic environments.
5. **Model Adaptation and Improvement (Future Work):**
In the future, **model adaptation** through retraining or updating based on drift alerts would close the lifecycle, ensuring accuracy as data distributions evolve in real-time systems.

Challenges and Solutions

The project also highlighted challenges in working with **Windows environments**, particularly in configuring **Kubernetes** and **Kubeflow**. These challenges required workarounds such as **WSL 2 (Windows Subsystem for Linux 2)** and **Minikube** but added complexity and delayed development. In future projects, a **Linux-based environment** or **cloud-based Kubernetes services** would provide a more seamless experience and better performance.

Project Impact and Broader Implications

This project demonstrated the potential of operationalizing deep learning models in real-world applications, specifically in improving urban mobility through traffic predictions. By establishing a robust deployment and monitoring pipeline, the project provided a scalable solution that could be extended beyond traffic prediction to other time-series forecasting tasks in smart city systems. The tools and techniques used—Docker for containerization, Kubernetes for scalability, and Evidently AI for monitoring—apply to any machine learning model requiring real-time predictions and continuous performance tracking.

Future Work

1. **Hyperparameter Tuning and Model Optimization**
While the LSTM model performed well, there is still room for improvement. Future work could focus on **hyperparameter optimization** and experimenting with advanced architectures like **Attention Mechanisms** or **Transformers**. These enhancements could lead to better accuracy and faster response times.
2. **Real-Time Data Integration**
Integrating the model with real-time data streams such as **Kafka** or **Apache Flink** would allow the model to continuously adapt to changing traffic conditions. This would

significantly improve the model's real-time application in smart city systems, enabling real-time retraining or updates as new data comes in.

3. **Ensemble Models and Comparative Analysis**

Future projects could explore deploying **ensemble models** that combine predictions from multiple models (e.g., LSTM, GRU, Transformer models). Comparative analysis of these models would offer deeper insights into their strengths and weaknesses under different traffic conditions.

4. **Cloud-Based Deployment for Global Scale**

While Kubernetes provided local scalability, future projects could focus on deploying the models in a **cloud-based infrastructure** such as **Google Kubernetes Engine (GKE)**, **Amazon EKS**, or **Azure AKS**. Cloud deployment would enable the model to serve predictions across multiple regions, allowing the system to operate on a global scale.

Closing Thoughts

In summary, the **Holistic Traffic Prediction for Smart Cities** project successfully demonstrated the full end-to-end lifecycle of a machine learning model, from **data collection and preprocessing**, through **model experimentation**, to **real-time deployment and monitoring**. The deployment of the LSTM model using Docker and Kubernetes ensured scalability and reliability, while **Evidently AI** provided continuous monitoring to maintain the model's performance over time. This project lays a strong foundation for future advancements in smart city technology, highlighting the potential of machine learning models to improve urban mobility and tackle real-time prediction challenges.

Team Task Breakdown

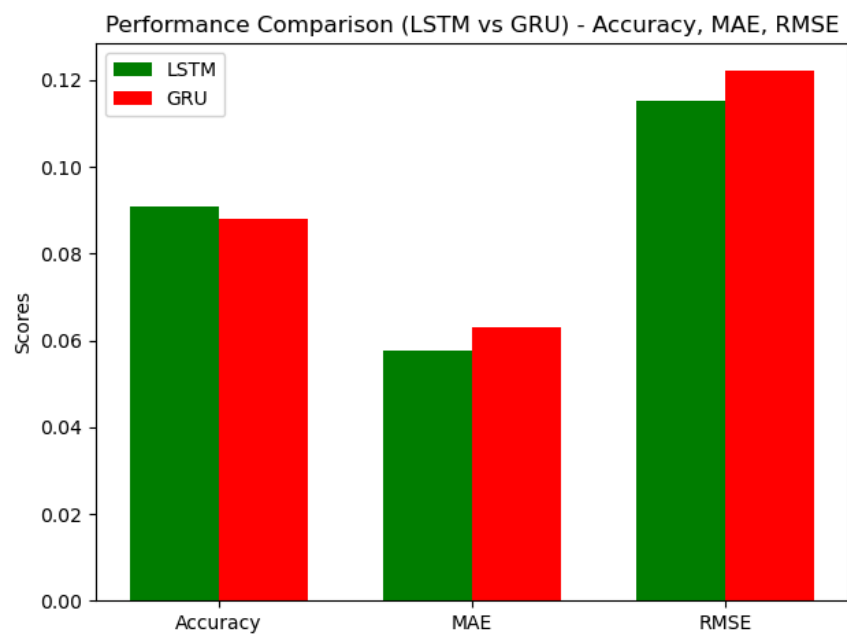
- **Peter Muller** (pmuller@andrew.cmu.edu): Team Lead, presenter, and responsible for QA/QC of reports and deliverables. Evidently Lead.
- **Ariana Rocha** (afrocha@andrew.cmu.edu): Responsible for compiling reports & presentations and ensuring deliverables are created as well as submitted on time.
- **Goutam Mukku** (gmukku@andrew.cmu.edu): GitHub manager and lead on model development, responsible for model experimentation and analysis.
- **Ayush Tripathi** (ayushit@andrew.cmu.edu): Manages Kubeflow experiment tracking and configuration, assisted in Kubeflow/Docker deployment
- **Katie Burgess** (keburgess@andrew.cmu.edu): Made presentation/presenter, assisted in submitting deliverables, and report writing/troubleshooting technical issues.

(Harshit Nanda, harshitn@andrew.cmu.edu: Teaching Assistant and advisor for the project, providing essential guidance and support. Honorary team member. Cookie Enjoyer. Friend.)

Appendices

1. **Appendix 1:** [model_experimentation_with_kubeflow_metrics.ipynb](#)
2. **Appendix 2:** [kubeflow_logs.pdf](#)
3. **Appendix 3:** [model_comparison_report.pdf](#)
4. **Appendix 4:** [docker_image_link.txt](#)
5. **Appendix 5:** [kubernetes_deployment_files.zip](#)
6. **Appendix 6:** [api_documentation.pdf](#)
7. **Appendix 7:** Evidently AI monitoring setup or link
8. **Appendix 8:** [report.html](#) (Evidently-generated drift report)
9. **Appendix 9:** AI Deployment Canvas
10. **Appendix 10:** AI ROI and Roadmap Canvas

Image 1:



94-879: Operationalizing AI - Team 6

GitHub Link: https://github.com/gmukku/AI_Ops_Project

Image 2:

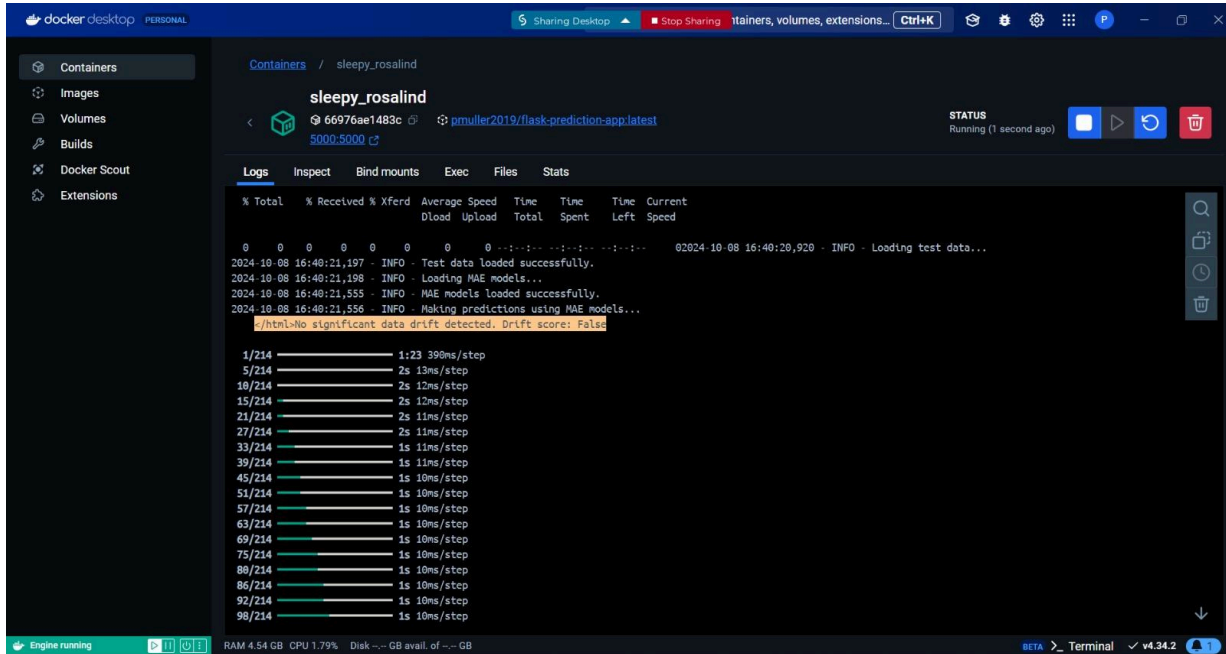
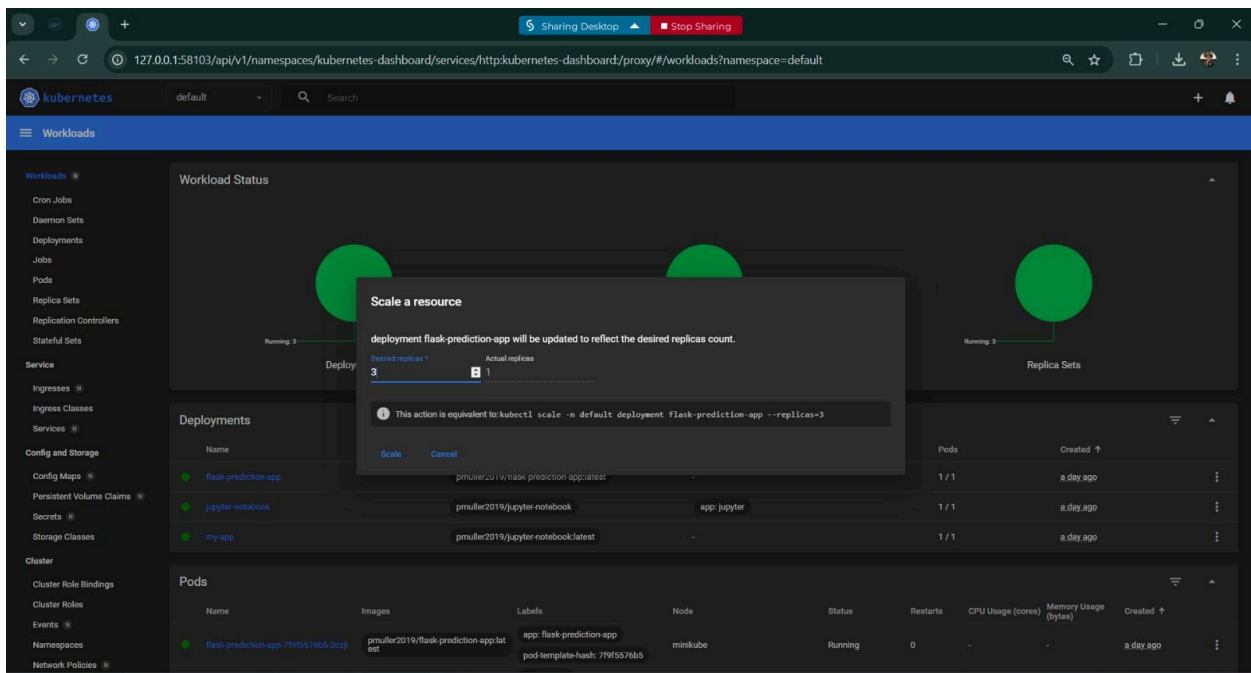


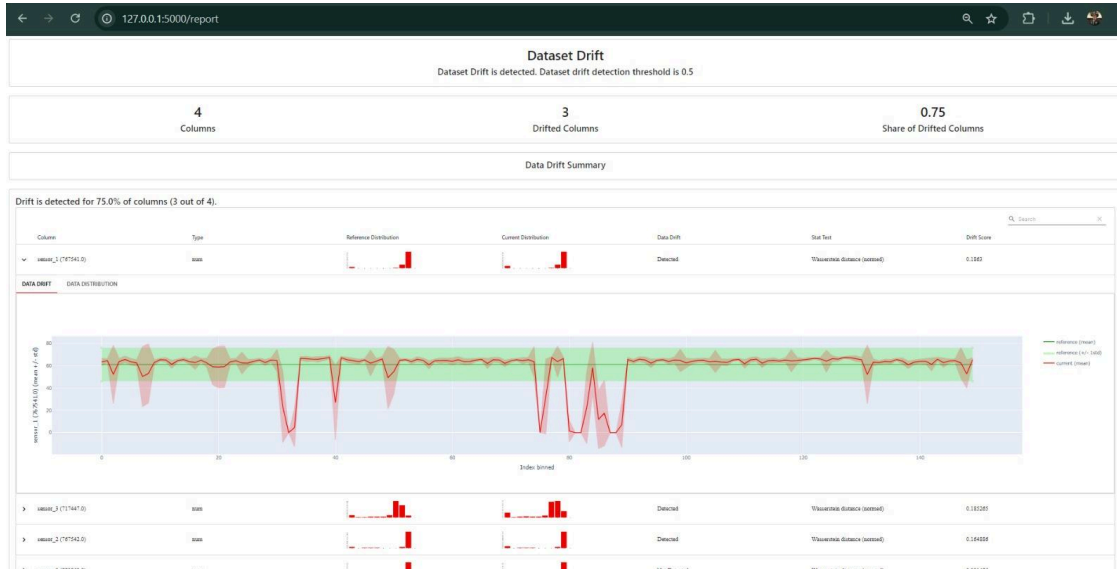
Image 3:



94-879: Operationalizing AI - Team 6

GitHub Link: https://github.com/gmukku/AI_Ops_Project

Image 4:



Appendix 9: AI Deployment Canvas

AI Deployment Canvas

Deployment Objectives:

- **Objective:** Deploy the LSTM model for real-time traffic flow predictions in a scalable, reliable manner.
 - **KPIs:**
 - Prediction accuracy (measured by MAE and RMSE).
 - Traffic management improvements (e.g., reduced congestion).
 - Model uptime and response time (scalability and efficiency).
-

Deployment Environment Setup:

- **Tools:**
 - **Docker:** For containerization and consistent deployment.
 - **Kubernetes:** For scalability, autoscaling pods, and orchestration.
 - **Flask API:** To expose prediction endpoints.
 - **Validation:** Tested through Kubernetes deployments, performance testing, and stress testing via Evidently AI for model monitoring.
 - **Infrastructure:** Running locally in **Minikube**, but designed for future scalability in cloud-based Kubernetes clusters (e.g., GKE or EKS).
-

Business Process Integration:

- **Integration:** Real-time traffic predictions fed into city planning systems.
 - **Business Impact:** Provides accurate forecasts to optimize traffic management and reduce congestion.
 - **Automation:** Automated alerts for significant traffic predictions, integrating with existing traffic control systems.
 - **Future Scope:** Scaling to other cities or regions, integrating predictions with autonomous vehicle systems.
-

Prediction Architecture:

- **Real-Time Architecture:**
 - Real-time traffic prediction based on streaming architecture.
 - Horizontal Pod Autoscaling enabled to adjust resource allocation based on traffic loads.
 - **Latency Requirement:** Sub-second response times for real-time predictions using Flask API.
-

Validation & Stress Testing:

- **Data Sources:** Historical traffic data from the **METR-LA dataset**.
 - **Validation:**
 - Tested with test datasets and pre-loaded sequences in Kubernetes.
 - **Evidently AI** is used to monitor performance and detect data drift.
 - Simulated traffic data used to validate robustness under high traffic loads.
 - **Stress Testing:** Conducted stress tests by increasing request loads and scaling Kubernetes pods to ensure model performance under peak demand.
-

Monitoring, Maintenance & Continual Learning:

- **Monitoring:**
 - Continuous monitoring using **Evidently AI** for model drift detection.
 - Automated alerts for data drift exceeding thresholds.
 - **Retraining:**
 - Model retraining protocols are in place for when drift is detected, ensuring model accuracy.
 - Future integration with real-time data streams to continuously update the model.
-

Risk Mitigation and Contingency Plans:

- **Technical Risks:**
 - Model drift due to changing traffic patterns (mitigated by Evidently AI alerts).
 - Infrastructure bottlenecks during high traffic (mitigated by Kubernetes autoscaling).
 - **Operational Risks:**
 - Deployment issues in Windows environments (mitigated by testing in Linux-based cloud environments).
 - Downtime risk mitigated by Kubernetes failover mechanisms.
 - **Compliance Risks:** Addressed data privacy concerns with anonymized traffic data.
-

Stakeholder Alignment & Training:

- **Key Stakeholders:**
 - City traffic managers.
 - Developers maintain the AI system.
 - Data scientists monitoring model performance.
- **Training:**

- Training sessions for stakeholders on using the traffic prediction dashboard and API.
- Clear documentation provided on API usage and model interpretation.
- Ongoing support for troubleshooting and updates.

Transition to BAU & Change Management:

- **Transition Plan:**

- Develop procedures for ongoing maintenance (model retraining, infrastructure scaling).
- Plan for continuous feedback and integration of new data sources.
- Ensure long-term support for model evolution, such as integrating future data streams or scaling across multiple cities.

AI Deployment Canvas Template below used as reference for above structure.

AI Deployment Canvas

AI Deployment Canvas		Name	Designed by:	Designed For:	Date:	Version:
Deployment Objectives - Define the goals for deploying the AI model, including business outcomes, operational objectives, and key performance indicators (KPIs) for measuring success.	Deployment Environment Setup - Specify the infrastructure (cloud/on-premise) and tools (e.g., Docker, Kubernetes). Conduct validation and testing to ensure performance, security, and scalability readiness.	Business Process Integration - Plan the integration of the AI model into existing business workflows, systems, APIs, and data pipelines to ensure usability and value creation.		Prediction Architecture - Select between batch, real-time, or streaming architectures, considering performance, scalability, and latency requirements.	Validation & Stress Testing - Specify the data needed, including sources, quality, and quantity, as well as technical infrastructure requirements to support the AI solution. Include the need for data that is representative and free from bias to ensure fairness in automation or augmentation.	
Monitoring, Maintenance & Continual Learning - Set up real-time monitoring for model performance, drift detection, and retraining protocols to keep the model up to date with business needs and data changes.		Risk Mitigation and Contingency Plans - Identify potential risks (technical, operational, compliance), and develop strategies to mitigate those risks during and after deployment.				
Stakeholder Alignment & Training - Engage relevant stakeholders, provide training, and assign roles to support adoption and operational efficiency post-deployment.		Transition to BAU & Change Management - Develop a comprehensive plan for transitioning the AI model to BAU, ensuring long-term sustainability and ongoing operational support.				

Created by Prof. Anand Rao (anand2@andrew.cmu.edu)

Appendix 10: AI ROI and Roadmap Canvas

AI ROI and Roadmap Canvas

Objectives:

- **Objective:** Improve urban mobility through real-time traffic predictions, optimizing traffic flow, reducing congestion, and minimizing emissions across Los Angeles, with the potential to scale to other cities.
 - **Strategic Alignment:** Aligns with broader objectives of smart city initiatives by improving infrastructure efficiency, reducing environmental impact, and enhancing the quality of urban life.
-

Inputs:

- **Hard Costs:**
 - **Hardware:** Servers for model deployment and data storage, GPU resources for training.
 - **Software:** Docker, Kubernetes, TensorFlow, Evidently AI, and Flask.
 - **Data:** METR-LA traffic dataset, additional real-time data streams (planned for future phases).
 - **Soft Costs:**
 - **Training:** Training team members on Kubeflow, Kubernetes, and Evidently AI.
 - **Change Management:** Integrating AI-based predictions into existing city traffic systems.
-

Risks:

- **Technical Risks:**
 - Model drift impacting prediction accuracy (mitigated by Evidently AI).
 - Scalability challenges in Windows environments (mitigated by using cloud services for Kubernetes).
 - **Operational Risks:**
 - Integration challenges with existing traffic systems.
 - Privacy concerns regarding real-time traffic data.
 - **Mitigation:** Continuous monitoring, regular retraining, and compliance with data privacy regulations.
-

Costs:

- **Hard Costs:**
 - Infrastructure (cloud-based deployment, server costs).
 - Licensing for Kubernetes and associated tools.
- **Soft Costs:**

- Ongoing training for team members.
- Change management processes for integrating the AI system into city operations.

Impacts:

- **Hard Benefits:**
 - Reduced traffic congestion, resulting in fuel cost savings for commuters.
 - Fewer road maintenance costs due to optimized traffic flow.
 - Enhanced productivity due to reduced travel times.
- **Soft Benefits:**
 - Improved air quality due to reduced emissions.
 - Better quality of life for residents due to smoother traffic conditions.
 - Enhanced the reputation of the city for using cutting-edge technology to address mobility issues.

Capabilities:

- **Technological Capabilities:**
 - Expertise in Kubernetes, Docker, TensorFlow, and AI model monitoring (Evidently AI).
 - Skills in developing and deploying time-series forecasting models like LSTM and GRU.
 - Strong capabilities in data science, AI model monitoring, and API development.
- **Operational Capabilities:**
 - Ability to scale AI systems across different urban regions.
 - Capability to retrain models based on real-time data streams and feedback loops.

Timeline & Milestones:

1. **Phase 1:**
 - Model development and experimentation with LSTM and GRU models.
 - Success criteria: Selection of best-performing model (completed).
2. **Phase 2:**
 - Containerization and Kubernetes deployment.
 - Success criteria: Successfully deployed LSTM model for real-time traffic predictions (completed).
3. **Phase 3:**
 - Implement monitoring with Evidently AI.

- Success criteria: Real-time monitoring in place, data drift detection functioning (in progress).

4. **Future Milestone:**

- Integration with real-time data streams and scaling to other cities.
- Success criteria: Real-time data integration and scaled deployment to multiple urban regions.

Benefits:

- **Quantified Hard Benefits:**

- Estimated reduction in traffic congestion by 10-15%.
- Potential fuel savings of approximately \$500K per year.

- **Soft Benefits:**

- Improved city reputation for using AI in public service.
- Greater public satisfaction with smoother traffic and fewer delays.
- Environmental benefits from reduced emissions and better traffic flow.










Portfolio Return on Investment:

- **ROI Analysis:**

- Initial investment costs in infrastructure and development are offset by long-term benefits such as fuel savings, improved productivity, and reduced congestion.
- **High ROI potential:** The benefits of optimized traffic flow and scalability to other cities justify the project's high initial costs.
- **Feasibility:** With the current success of Phase 2, scaling the project is both feasible and cost-effective, providing additional future value.

AI ROI and Roadmap Canvas Template below used as reference for above structure.

AI ROI and Roadmap Canvas

AI ROI and Roadmap Canvas		Name	Designed by:	Designed For:	Date:	Version:
Objectives  <Clearly define the strategic goals of the AI initiative, aligning with broader business objectives. Specify the purpose of the portfolio, distinguishing between initiatives aimed at staying in business, generating ROI, and creating future options>						
Inputs  <List the necessary resources, categorized into hard costs (e.g., hardware, software, data) and soft costs (e.g., training, change management), ensuring that all financial, human, and technological inputs are accounted for>	Impacts  <Detail the anticipated impacts of the AI initiative from individual, organizational, and societal perspectives. Include metrics for both hard benefits (e.g., time savings) and soft benefits (e.g., improved decision-making)>	Timeline & Milestones  <Outline the project phases, key deliverables, and deadlines, incorporating checkpoints for evaluating the realization of hard and soft benefits. Ensure alignment with prioritized initiatives and strategic objectives>				
Risks  <Identify potential risks associated with the AI project, categorizing them into consumer, company, societal, and environmental risks. Include strategies for mitigating these risks and account for both the direct and indirect costs of risk management>	Capabilities  <Specify the skills, expertise, and technological capabilities required to successfully develop, deploy, and manage the AI solution. Ensure that the capabilities align with the strategic objectives and are sufficient to deliver both hard and soft benefits>					
Costs  <Provide a detailed breakdown of the financial expenditure required for the AI project, including both hard costs (e.g., infrastructure, licensing) and soft costs (e.g., employee training, compliance). Allocate these costs across different phases of the project and categorize them according to their relevance to stay-in-business, ROI-generating, or option-creating initiatives>		Benefits  <Quantify the expected returns from the AI project, detailing both hard benefits (e.g., cost savings, revenue growth) and soft benefits (e.g., brand reputation, employee satisfaction). Ensure that these benefits align with the organization's strategic objectives and provide a clear value proposition>				
Portfolio Return on Investment  <Evaluate the overall ROI for the AI portfolio, identifying the proportion of initiatives aimed at staying in business, generating ROI, and creating future options. Use a value-effort matrix to prioritize initiatives with the highest impact and feasibility, ensuring a balanced portfolio approach>						