SciTec MLOps Engineering Coding Challenge Instructions

Contents

1. Problem Statement

You are tasked with designing and demonstrating a machine learning solution for classifying missile flight phases. Your implementation will involve building an LSTM model for classification, deploying it in a containerized environment, and following MLOps best practices for operationalization. The solution should emphasize scalability, robustness, and adherence to modern MLOps standards.

Your task can be summarized as follows:

- Develop a robust LSTM model to classify missile track segments into reentry and non-reentry phases.
- Deploy your solution using Docker and Kubernetes (Minikube).
- Demonstrate scalability, logging, and monitoring capabilities.

2. Context

Missile Tracking and Classification

- Simulated sensors track ballistic missiles and record telemetry data.
- The dataset includes position (latitude, longitude, altitude), radiometric intensity, and timestamps.
- Your goal is to classify whether a missile is in the reentry phase using labeled training data.

Deployment Requirements

- The model should be deployable in a containerized environment using Kubernetes (Minikube) for orchestration.
- The system must log inference times and predictions for monitoring.

3. Input Data

Files

The following files are provided:

- `train.csv`: Labeled data for training.
    - Columns: `timestamp`, `track_id`, `sensor_id`, `latitude`, `longitude`, `altitude`, `radiometric_intensity`, `reentry_phase`.
- `test.csv`: Unlabeled data for testing.
    - Same columns as `train.csv`, excluding `reentry_phase`.

Notes

- Sensor data includes noise to simulate real-world conditions.
- Preprocessing and feature engineering are encouraged to improve model performance.

4. Requirements

Languages and Tools

- Use Python for implementation.
- Use TensorFlow or PyTorch for model development.

Machine Learning

- Build an LSTM model for classification.

- Preprocess data:
  - Normalize features.
  - Handle noisy data.
- Use regularization techniques (e.g., dropout, batch normalization) and callbacks (e.g., early stopping).
- Evaluate the model against a simple baseline classifier (e.g., altitude threshold).

Deployment and Operations

- Containerize the solution using Docker.
- Deploy the containerized model to Minikube using Kubernetes manifests.
- Include logging to capture inference times and predictions.

Scalability and Testing

- Ensure the solution can scale to handle larger datasets or higher inference loads.
- Provide unit tests and coverage reports to validate functionality.

5. Allowed Resources

- Use any free and open-source software.
- Consult external resources with proper citations.

6. What to Deliver

Code

- Modular Python scripts, structured for production.
- Include unit tests and coverage reporting.

Documentation

- Instructions for running the solution locally and in Minikube.
- A detailed README.md describing:
  - Model architecture and rationale.
  - Data preprocessing and feature engineering steps.
  - Scalability considerations and trade-offs.
  - Comparison to baseline performance.

Deployment

- Dockerfile and Kubernetes manifests for deployment.

- Logs from deployed inference runs.

Performance Validation

- Metrics for model evaluation and inference times.
- Description of scalability testing methodology.

7. Assessment Criteria

Scores for each category will be weighted according to the percentages in **Table 1**. Final scores are a weighted sum of technical skills, decision-making, and creativity.

8. Test Environment

Your solution will be tested on a Linux system with:

- **CPU**: 24 CPUs @ 3.5 GHz
- **RAM**: 64 GB
- **Storage**: 1 TB
- **GPU**: NVIDIA RTX A2000
- **OS**: RHEL 9.5

Table 1. Coding Challenge Evaluation Criteria

Category	Weight	Evaluation Criteria
Model Implementation	30%	Completeness, correctness, and adherence to ML best practices, including preprocessing and evaluation.
Deployment and Ops	30%	Correctness and efficiency of containerization, deployment, and logging.
Code Quality	20%	Modularity, documentation, and adherence to Python best practices.
Performance Evaluation	10%	Use of appropriate metrics and comparison to a baseline.
Scalability and Testing	10%	Demonstration of scalability and robust testing practices.