COLLEGE CODE: 1133

COLLEGE NAME: VELAMMAL INSTITUTE OF TECHNOLOGY

DEPARTMENT: ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

STUDENT NM-ID: aut113323aib47

ROLL NO: 113323243094

DATE: 02.05.2025

TECHNOLOGY-ROOT CAUSE ANALYSIS FOR EQUIPMENT FAILURES

SUBMITTED BY,

SAMYUKTHA SELVARAJ

7010597182

PHASE 4: Performance of the project

Title: Root Cause Analysis For Equipment Failures

Objective:

To systematically identify the underlying causes of equipment failures in order to implement effective corrective and preventive actions, reduce downtime, improve equipment reliability, and enhance overall operational efficiency.

1.AI Model Performance Enhancement:

Overview:

AI model performance enhancement involves strategies and techniques used to improve the accuracy, efficiency, and generalization of machine learning (ML) or deep learning (DL) models. Enhancing performance is crucial for ensuring models produce reliable and meaningful predictions, especially in real-world applications like healthcare, finance, and autonomous systems.

Performance Improvement Techniques

1. **Data Quality and Quantity**
    - Clean, relevant, and well-labeled data increases model accuracy.
    - Data augmentation (for images, text) helps models generalize better.
    - Synthetic data generation in case of imbalanced datasets.
2. **Feature Engineering**
    - Selecting meaningful features and reducing noise boosts learning efficiency.
    - Dimensionality reduction (PCA, t-SNE) can enhance model speed and accuracy.
3. **Algorithm Optimization**
    - Choosing better-performing models for specific tasks (e.g., XGBoost for tabular data, Transformers for NLP).
    - Hyperparameter tuning (grid search, random search, Bayesian optimization).

Outcome

- **Improved Accuracy and Precision**: The model makes more correct predictions.
- **Generalization to Unseen Data**: Better performance on real-world or test data.

2.Chatbot Performance Optimization:

Overview

Chatbot performance optimization refers to enhancing the effectiveness, accuracy, responsiveness, and user experience of AI-powered conversational agents. This includes refining both the backend (e.g., natural language understanding and generation) and frontend (e.g., UI, integration) to ensure smooth, intelligent, and human-like interactions with users.

Performance Improvement Techniques

1. **Natural Language Processing (NLP) Enhancements**
    - Use of advanced NLP models (like GPT, BERT, T5) to improve understanding and response generation.
    - Incorporating named entity recognition (NER), intent classification, and sentiment analysis.
2. **Context Management**
    - Implement memory or session tracking to maintain coherent conversations over multiple turns.
    - Use context-aware models to personalize and humanize interactions.
3. **Training on Domain-Specific Data**
    - Fine-tuning on company-specific or industry-specific datasets to improve relevance and accuracy.
    - Including FAQs, chat logs, and support tickets for realistic scenarios.

Outcome

- **Higher Accuracy and Relevance**: More accurate understanding of user intents and better responses.

- **Improved User Satisfaction**: Enhanced user experience through natural and context-aware conversations.
- **Faster Response Time**: Real-time or near-instant replies for better engagement.

3.IoT Integration Performance:

Overview

IoT Integration Performance refers to how effectively Internet of Things (IoT) devices and systems communicate, coordinate, and deliver real-time data insights across platforms and applications. High-performance integration ensures seamless connectivity, minimal latency, robust data handling, and secure communication among a network of smart devices, sensors, and cloud services.

Performance Improvement Techniques

1. **Network Optimization**
    - Use low-latency and high-bandwidth communication protocols (e.g., MQTT, CoAP, 5G).
    - Implement edge computing to reduce data transmission delays and offload cloud processing.
2. **Efficient Data Handling**
    - Use data compression, aggregation, and filtering to reduce the volume of transmitted data.
    - Implement real-time data streaming and batch processing intelligently.

Outcome

- **Real-Time Responsiveness**: Faster and more reliable communication between devices and platforms.
- **Improved System Reliability**: Reduced downtime and increased fault tolerance across the network.
- **Efficient Resource Utilization**: Optimized bandwidth and energy usage through smart data processing.

4.Data Security and Privacy performance:

Overview

Data Security and Privacy Performance refers to how effectively a system protects sensitive data from unauthorized access, breaches, and misuse while ensuring compliance with data protection regulations. It measures the strength and efficiency of the mechanisms used to secure data during storage, processing, and transmission, as well as how well user privacy is preserved.

Performance Improvement Techniques

1. **Encryption and Tokenization**
    - Use strong encryption protocols (AES-256, RSA) for data at rest and in transit.
    - Tokenize sensitive information (like credit card numbers) to reduce risk.
2. **Access Control and Authentication**
    - Implement multi-factor authentication (MFA) and role-based access control (RBAC).
    - Use identity and access management (IAM) systems to govern user privileges.

Outcome

- **Reduced Risk of Data Breaches**: Strong security measures prevent unauthorized access.
- **Improved Regulatory Compliance**: Avoid penalties and maintain trust by meeting legal obligations.

5.Performance Testing And Metrics Collection:

Overview

Performance Testing and Metrics Collection involves evaluating a system's speed, responsiveness, stability, and scalability under expected or stress conditions. It helps identify bottlenecks, weaknesses, and areas for optimization. Metrics collection provides quantitative data to support performance decisions and ongoing monitoring.

Performance Improvement Techniques

1. **Types of Performance Testing**
    - **Load Testing**: Measures how the system handles expected user traffic.
    - **Stress Testing**: Evaluates system behavior under extreme conditions.
    - **Spike Testing**: Tests sudden increases in load.
    - **Endurance Testing**: Checks for memory leaks and stability over time.
    - **Scalability Testing**: Assesses system performance as demand increases.
2. **Automation Tools**
    - Use tools like JMeter, LoadRunner, Gatling, or Locust for repeatable and scalable test execution.
    - Integrate performance tests into CI/CD pipelines for continuous evaluation.

Outcome

- **Faster Application Response Times**: Enhances user experience and engagement.
- **Improved System Stability and Scalability**: Handles more users and data without degradation.
- **Early Detection of Bottlenecks**: Fix issues before they affect users in production.

Key Challenges in Phase 4:

1.Scaling for high-volume failures:

Challenge: Systems crash or degrade under sudden high traffic due to limited resources and poor fault isolation.
Solution: Implement auto-scaling, load balancing, caching, and microservices with robust monitoring and failure recovery strategies.

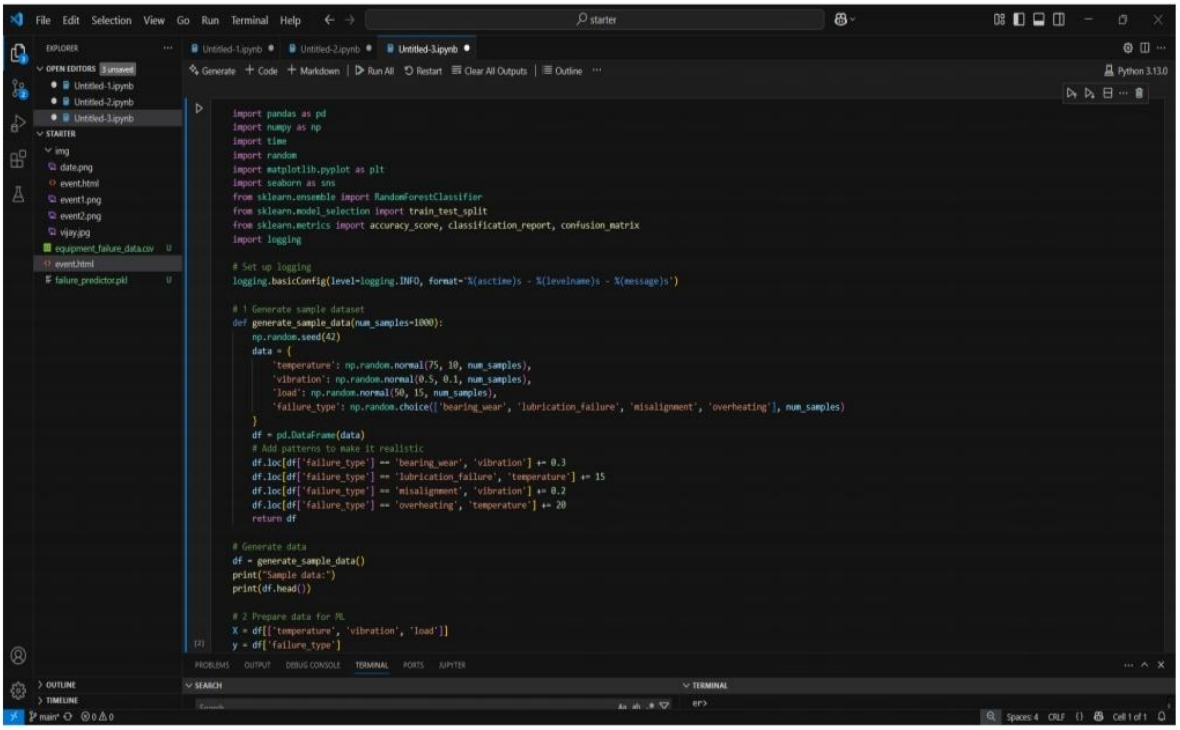
2.OEM data silos:

Challenge: OEM data silos prevent seamless data sharing across departments, partners, or systems, leading to inefficiencies and poor decision-making.
Solution: Use data integration platforms, APIs, and standardized data formats to unify siloed data into a centralized or interoperable system.

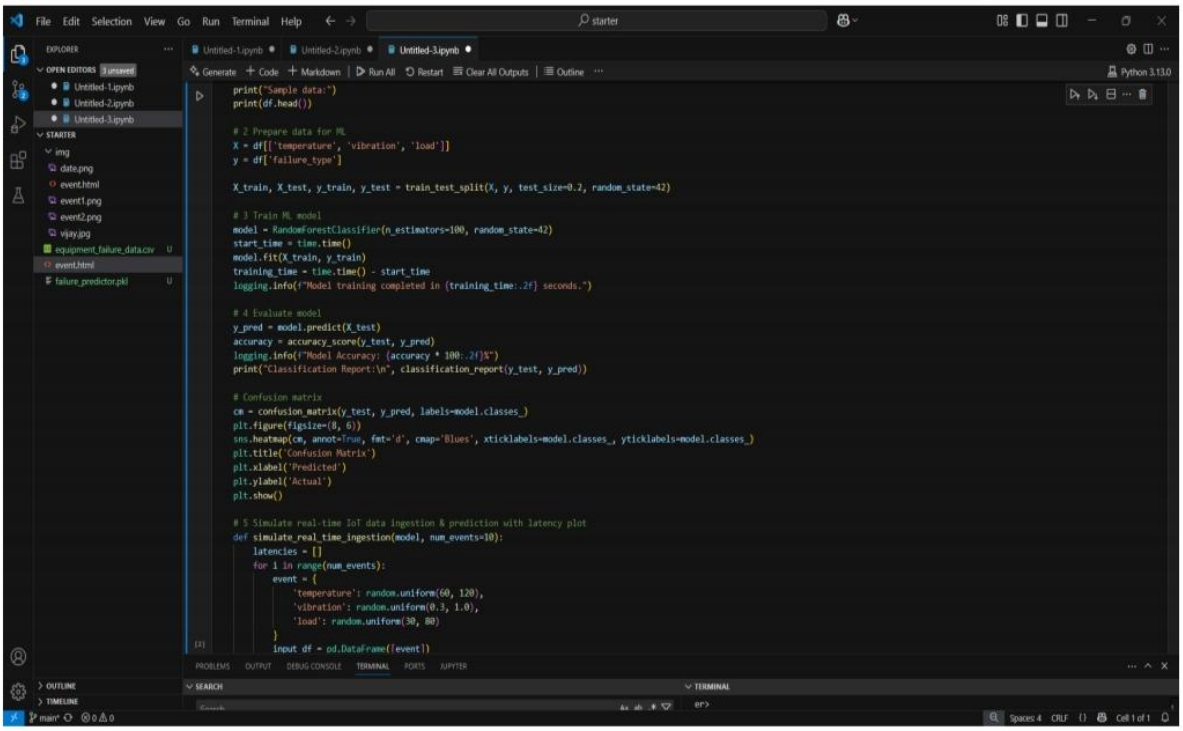
3.False positives in predictive RCA:

Challenge: False positives in predictive Root Cause Analysis (RCA) lead to misdiagnosed issues, wasted resources, and reduced trust in the system.
Solution: Improve model accuracy with high-quality labeled data, apply advanced anomaly detection, and incorporate human-in-the-loop validation for critical decisions.

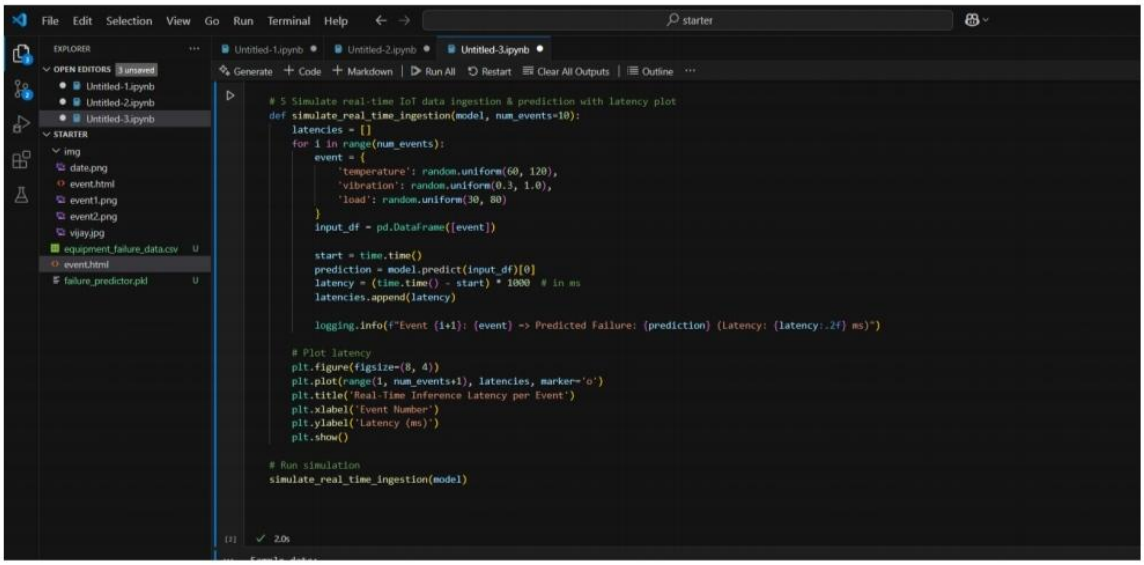
Sample code for phase 4:



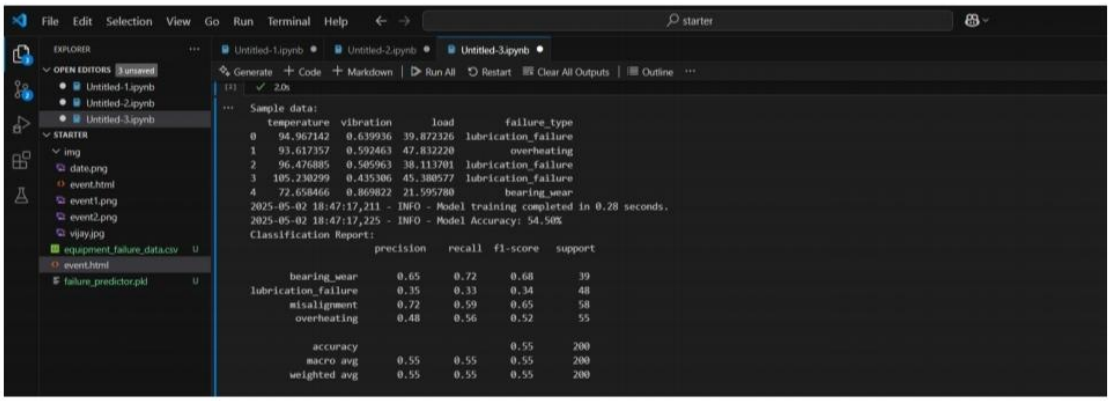
```
import pandas as pd
import numpy as np
import time
import random
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import logging

# Set up logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

# 1 Generate sample dataset
def generate_sample_data(num_samples=1000):
    np.random.seed(42)
    data = {
        'temperature': np.random.normal(75, 10, num_samples),
        'vibration': np.random.normal(0.5, 0.1, num_samples),
        'load': np.random.normal(50, 15, num_samples),
        'failure_type': np.random.choice(['bearing_wear', 'lubrication_failure', 'misalignment', 'overheating'], num_samples)
    }
    df = pd.DataFrame(data)
    # Add patterns to make it realistic
    df.loc[df['failure_type'] == 'bearing_wear', 'vibration'] += 0.3
    df.loc[df['failure_type'] == 'lubrication_failure', 'temperature'] += 15
    df.loc[df['failure_type'] == 'misalignment', 'vibration'] += 0.2
    df.loc[df['failure_type'] == 'overheating', 'temperature'] += 20
    return df

# Generate data
df = generate_sample_data()
print("Sample data:")
print(df.head())

# 2 Prepare data for ML
X = df[['temperature', 'vibration', 'load']]
y = df['failure_type']
```

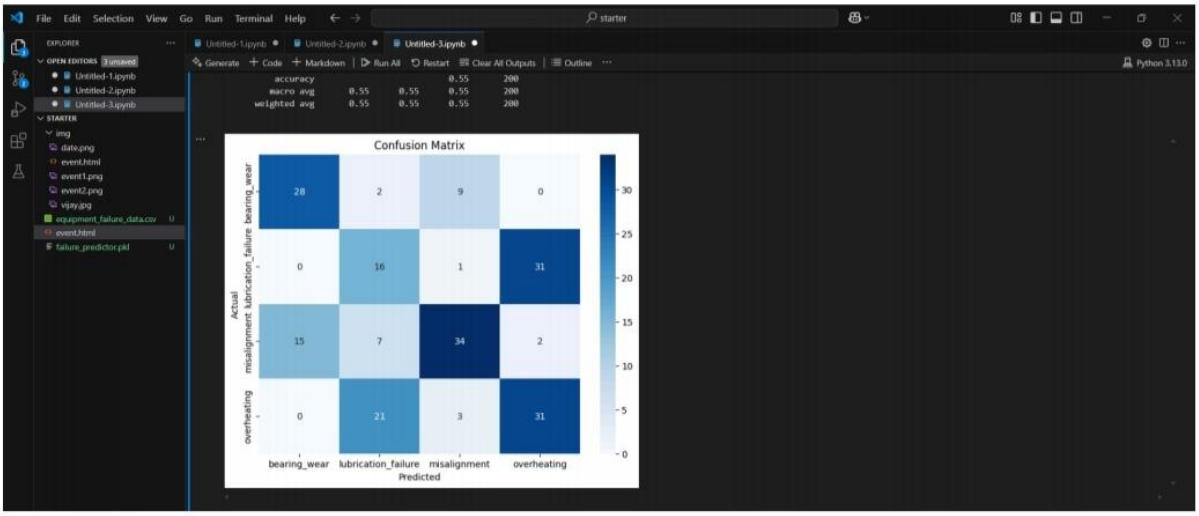


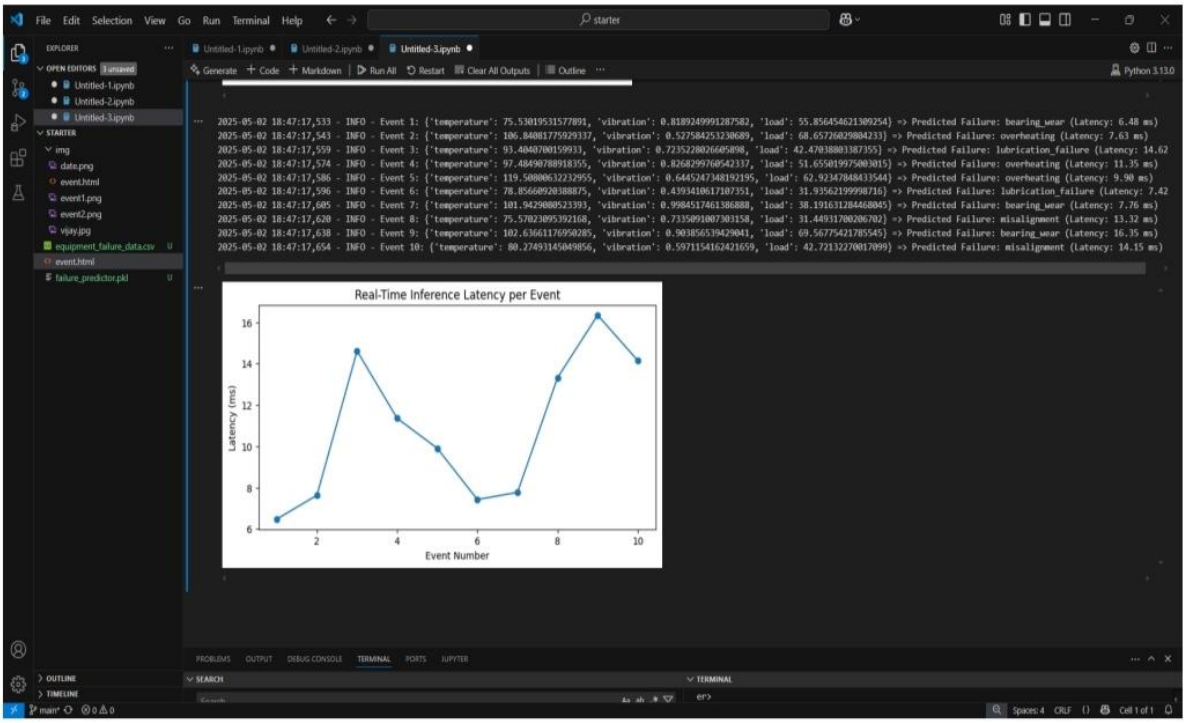
```
print("Sample data:")
print(df.head())

# 2 Prepare data for ML
X = df[['temperature', 'vibration', 'load']]
y = df['failure_type']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3 Train ML model
model = RandomForestClassifier(n_estimators=100, random_state=42)
start_time = time.time()
model.fit(X_train, y_train)
training_time = time.time() - start_time
logging.info(f"Model training completed in {training_time:.2f} seconds.")

# 4 Evaluate model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
logging.info(f"Model Accuracy: {accuracy * 100:.2f}%")
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred, labels=model.classes_)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes_, yticklabels=model.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# 5 Simulate real-time IoT data ingestion & prediction with latency plot
def simulate_real_time_ingestion(model, num_events=10):
    latencies = []
    for i in range(num_events):
        event = {
            'temperature': random.uniform(60, 120),
            'vibration': random.uniform(0.3, 1.0),
            'load': random.uniform(30, 80)
        }
        input_df = pd.DataFrame([event])
```



Performance Metrics Screenshot for Phase 4:




File Edit Selection View Go Run Terminal Help
starter
Untitled-1.ipynb
Untitled-2.ipynb
Untitled-3.ipynb
Generate Code Markdown Run All Restart Clear All Outputs Outline
Python 3.13.0
accuracy 0.55 200
macro avg 0.55 0.55 0.55 200
weighted avg 0.55 0.55 0.55 200
Confusion Matrix
Actual
Predicted
bearing_wear
lubrication_failure
misalignment
overheating
28 2 9 0
0 16 1 31
15 7 34 2
0 21 3 31


File Edit Selection View Go Run Terminal Help
starter
Untitled-1.ipynb
Untitled-2.ipynb
Untitled-3.ipynb
Generate Code Markdown Run All Restart Clear All Outputs Outline
Python 3.13.0
2025-05-02 18:47:17,533 - INFO - Event 1: {'temperature': 75.53019531577891, 'vibration': 0.8189249991287582, 'load': 55.856454621309254} => Predicted Failure: bearing_wear (Latency: 6.48 ms)
2025-05-02 18:47:17,543 - INFO - Event 2: {'temperature': 106.84081775929337, 'vibration': 0.527584253230689, 'load': 68.65726029804233} => Predicted Failure: overheating (Latency: 7.63 ms)
2025-05-02 18:47:17,559 - INFO - Event 3: {'temperature': 93.40407001599033, 'vibration': 0.7235228026605898, 'load': 42.47038803873755} => Predicted Failure: lubrication_failure (Latency: 14.62
2025-05-02 18:47:17,574 - INFO - Event 4: {'temperature': 97.48498788918355, 'vibration': 0.8268299760542337, 'load': 51.655019975003015} => Predicted Failure: overheating (Latency: 11.35 ms)
2025-05-02 18:47:17,586 - INFO - Event 5: {'temperature': 119.50800632232955, 'vibration': 0.6445247348192195, 'load': 62.92347048431544} => Predicted Failure: overheating (Latency: 9.90 ms)
2025-05-02 18:47:17,596 - INFO - Event 6: {'temperature': 78.85668920388875, 'vibration': 0.4393410617107351, 'load': 31.935621999998716} => Predicted Failure: lubrication_failure (Latency: 7.42
2025-05-02 18:47:17,605 - INFO - Event 7: {'temperature': 101.5429080523393, 'vibration': 0.9984517461386888, 'load': 38.191631284468045} => Predicted Failure: bearing_wear (Latency: 7.76 ms)
2025-05-02 18:47:17,620 - INFO - Event 8: {'temperature': 75.57023095392168, 'vibration': 0.7335091007301158, 'load': 31.449317000206702} => Predicted Failure: misalignment (Latency: 13.32 ms)
2025-05-02 18:47:17,638 - INFO - Event 9: {'temperature': 102.63661176950285, 'vibration': 0.9038565394290041, 'load': 69.56775421785545} => Predicted Failure: bearing_wear (Latency: 16.35 ms)
2025-05-02 18:47:17,654 - INFO - Event 10: {'temperature': 80.27493145049856, 'vibration': 0.5971154162421659, 'load': 42.72132270017099} => Predicted Failure: misalignment (Latency: 14.15 ms)
Real-Time Inference Latency per Event
Latency (ms)
Event Number
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER
SEARCH
TERMINAL