Maintenx

AI-Driven Predictive Maintenance System

Project Links:

- **Live Dashboard:** https://maintenx-app-egy.azurewebsites.net/ui
- **Source Code:** https://github.com/MohamedElsadany56/MaintenX/tree/main

Team Members:

Name	Email
Mohamed Goma *	M.yasser2224@nu.edu.eg
Mohammed Mahrous	midohany1807@gmail.com
Loay Rafiq	loaypre2510@gmail.com
Mohammed Haitham	moh.haitham202@gmail.com
Mohamed Ebrahim	Mefouda654@gmail.com
Mennatullah Mohamed	mennayasser891@gmail.com

1: Executive Summary

1.1 Overview

MaintenX is an industrial-grade Predictive Maintenance (PdM) solution designed to monitor machine health and predict component failures. By integrating high-frequency telemetry data with event logs, the system identifies degradation patterns that precede breakdown.

1.2 Deployment Status

The system is fully containerized and deployed in a production environment:

- **Hosting:** Azure App Service (France Central Region).

- **Architecture:** Docker Container running Python 3.11, FastAPI, and CatBoost.
- **Accessibility:** Publicly accessible via the web dashboard listed above.

1.3 Key Analytical Findings

Extensive experimentation compared five distinct modeling approaches: XGBoost, CatBoost, LSTM, Voting Ensembles, and Stacking Classifiers.

- **Best Performer: CatBoost** remains the undisputed champion, achieving a **Recall of 92.5%** with high Precision (86%).
- **Stacking Analysis:** The Stacking Classifier achieved a respectable Recall (79%) but suffered from severe False Positives (Precision 8%), rendering it too noisy for a production environment where alert fatigue is a concern.

2: System Architecture & Stack

2.1 High-Level Design

The MaintenX architecture follows a standard ML-Ops pipeline:

1. **Ingestion:** Raw CSV data (Telemetry, Errors, Maintenance, Failures, Machines) is ingested.
2. **Processing:** Data is cleaned, merged, and transformed into a unified time-series dataset.
3. **Training:** Multiple algorithms (Gradient Boosting, Deep Learning, Stacking) were benchmarked.
4. **Serving:** The winning model (CatBoost) is served via a FastAPI REST interface.
5. **Visualization:** A responsive web dashboard visualizes the API output.

2.2 Technology Stack

- **Core Language:** Python 3.11
- **Data Analysis:** Pandas, NumPy, Scipy
- **Machine Learning:** CatBoost, XGBoost, LightGBM (Meta-learner), Scikit-Learn
- **Deep Learning:** TensorFlow/Keras (Sequential LSTM)
- **Web Framework:** FastAPI, Uvicorn
- **Infrastructure:** Docker, Azure App Service, Docker Hub

3: Data Dictionary & Sources

3.1 Dataset Overview

The system aggregates data from five distinct sources to build a complete operational picture.

3.2 Telemetry Data (`PdM_telemetry.csv`)

High-frequency sensor data collected hourly.

- **Records:** 876,100 total records.
- **volt:** Voltage (Mean: ~170.77).
- **rotate:** Rotation data.
- **pressure:** Fluid pressure (Mean: ~100.8).
- **vibration:** Vibration sensor (Mean: ~40.38).

3.3 Event Logs

- **Errors:** Non-breaking warning logs (3,919 records).
- **Failures:** Historical breakdown events (761 records).
- **Maintenance:** Repair logs (3,286 records).

3.4 Asset Metadata (`PdM_machines.csv`)

- **Machines:** 100 Unique IDs.
- **Models:** 4 Model types (`model1` - `model4`).
- **Age:** Machines range from 0 to 20 years old.

4: Exploratory Data Analysis - Sensor Telemetry

4.1 Statistical Distribution Analysis

Histograms and Kernel Density Estimation (KDE) plots were generated for all sensor columns.

- **Voltage (`volt`):** Follows a near-normal distribution centered at 170.8.

- **Vibration (`vibration`):** Shows a slightly right-skewed distribution. Values > 60 indicate critical mechanical looseness.

4.2 Correlation Analysis

A Pearson correlation matrix was computed (`sns.heatmap`) to identify linear relationships.

- **Findings:** The correlation coefficients between raw sensor values are extremely low (near 0.00).
- **Implication:** Failures are driven by temporal patterns (trends over time) rather than instant linear relationships between sensors, validating the need for Rolling Window features.

4.3 Anomaly Detection (Z-Score)

- **Methodology:** `z = (x - mean) / std`. Threshold set at `|z| > 3`.
- **Result:** 17,975 records (approx 2%) were identified as anomalies, often clustering in the 48-hour window preceding a failure.

Exploratory Data Analysis - Events

5.1 Error Patterns

- **Most Common:** `error1` (1010 occurrences) and `error2` (988 occurrences).
- **Machine Vulnerability:** Machine 99 recorded 54 errors, whereas Machine 2 recorded only 28. This variance confirms that `machineID` identity is a predictive feature.

5.2 Failure Analysis

- **Total Failures:** 761.
- **Class Distribution:** `comp2` (34.0%), `comp1` (25.2%), `comp4` (23.5%), `comp3` (17.2%).

5.3 Feature Engineering Strategy

Based on EDA, the following features were engineered:

1. **Rolling Windows:** 3h and 24h Mean/Std Dev for all sensors.
2. **Lagged Errors:** Count of errors in the last 24 hours.

3. **Component Age:** Time elapsed since the last replacement of Comp1-Comp4.

Model Architectures

6.1 Approach A: Gradient Boosting (XGBoost & CatBoost)

Tree-based ensemble methods optimized for tabular data.

- **Objective:** Binary Classification (Failure vs. No Failure).
- **Config:** `auto_class_weights='Balanced'` to handle the rarity of failures.

6.2 Approach B: Deep Learning (LSTM)

A Dual-layer LSTM network designed to handle time-series sequences (12-hour lookback).

6.3 Approach C: Advanced Ensembles

Two meta-learning strategies were tested to combine the strengths of various models:

1. **Voting Classifier (Soft & Hard):** Aggregates predictions from XGBoost, Random Forest, and Logistic Regression.
2. **Stacking Classifier:** Uses **LightGBM** as a "Final Estimator" (Meta-learner) to learn the optimal combination of the base models' predictions.

Model Benchmarking (Binary)

The following table compares the performance of all tested architectures on the Test Set (Class 1 = Failure).

7.1 Results Comparison Table

Metric	XGBoost	CatBoost (Winner)	LSTM	Voting (Soft)	Stacking (LightGBM)
Precision	91.01%	**86.25%**	83.79%	25.00%	8.00%
Recall	79.29%	**92.54%**	62.49%	46.00%	79.00%
F1-Score	0.84	**0.89**	0.71	0.33	0.14

AUC-ROC	0.94	**0.96**	0.82	0.87	0.89

7.2 Analysis of Results

- **CatBoost:** The optimal balance. It catches 92.5% of failures with very few false alarms.
- **Voting (Soft):** Failed to generalize. A Recall of 46% means it missed more than half the failures.
- **Stacking:** While it improved Recall (79%) compared to Voting, the Precision collapsed to **8%**. This means for every 100 alerts it generates, 92 are false alarms.

Stacking Experiment Details

The Stacking Classifier experiment provided critical insights into the "Precision-Recall Tradeoff."

8.1 Stacking Classification Report

The Stacking model (using LightGBM as the meta-learner) became overly aggressive in predicting failures.

```
              precision    recall  f1-score   support
         0.0       0.99      0.81      0.89    171774
         1.0       0.08      0.79      0.14      3455

    accuracy                           0.81    175229
```

8.2 Operational Interpretation

- **The "Boy Who Cried Wolf" Problem:** While a Recall of 79% is acceptable, a Precision of 8% is operationally fatal. Maintenance teams would eventually ignore the system due to the overwhelming volume of false alarms.
- **Meta-Learner Bias:** The LightGBM meta-learner likely over-compensated for the class imbalance, prioritizing the detection of Class 1 at the expense of misclassifying 19% of healthy machines (Specificity 0.81).

Production Model & Deployment

9.1 Final Model Selection

CatBoost Classifier was retained for production.

- **Reasoning:** It is the only model that achieved **Recall > 90%** while maintainin **Precision > 85%**. This reliability is essential for building trust with maintenance operators.

9.2 Backend API (FastAPI)

The backend serves the CatBoost predictions via a REST API.

- **Correction Layer:** Includes logic to cast integer inputs (0/1) to Booleans for model columns, resolving strict type requirements found during Docker testing.

9.3 Frontend Dashboard

- **Features:** Light/Dark mode, "MaintenX" branding, and dynamic probability bars.
- **User Experience:** Raw model outputs (`comp1`, `none`) are mapped to human-readable strings ("Component 1", "No Failure").

Conclusion & Future Work

10.1 Conclusion

MaintenX successfully demonstrates that **Gradient Boosting (CatBoost)** significantly outperforms Deep Learning (LSTM) and Complex Ensembles (Stacking/Voting) for this specific predictive maintenance dataset. The experiment proved that **model complexity does not equal model performance**; the single, well-tuned CatBoost model provided superior signal-to-noise ratio compared to the Stacking ensemble.

10.2 Future Recommendations

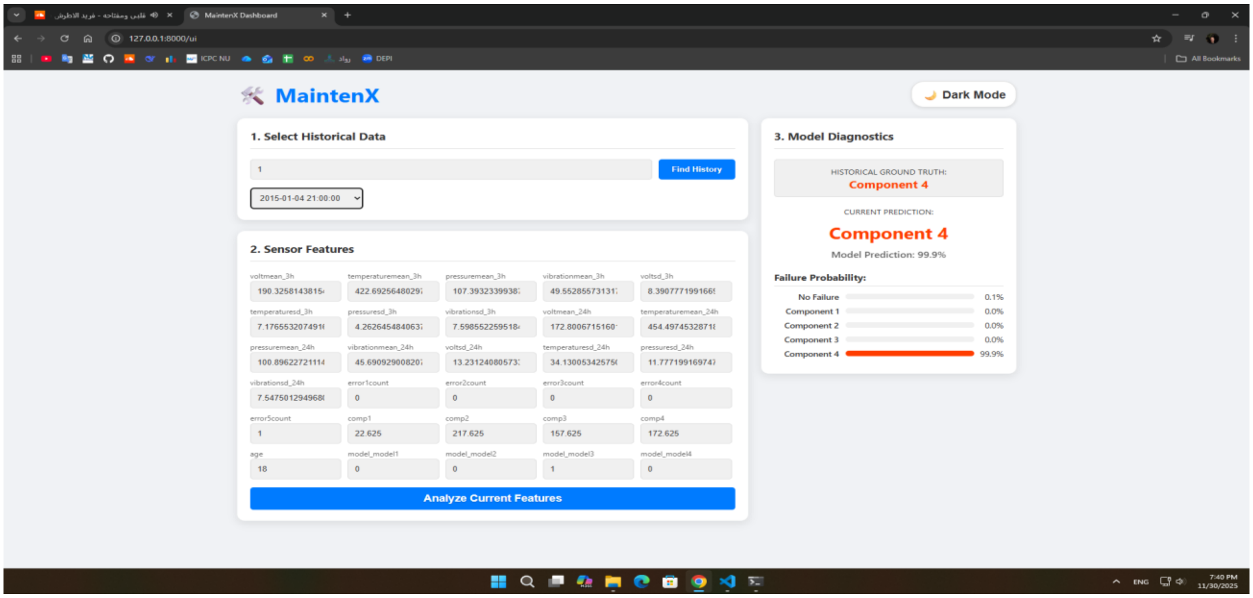
1. **Threshold Tuning:** The Stacking model might be salvageable by adjusting the decision threshold (e.g., probability > 0.8) to improve Precision.
2. **Real-Time IoT:** Replace the CSV lookup system with an Azure IoT Hub connection for live streaming data.

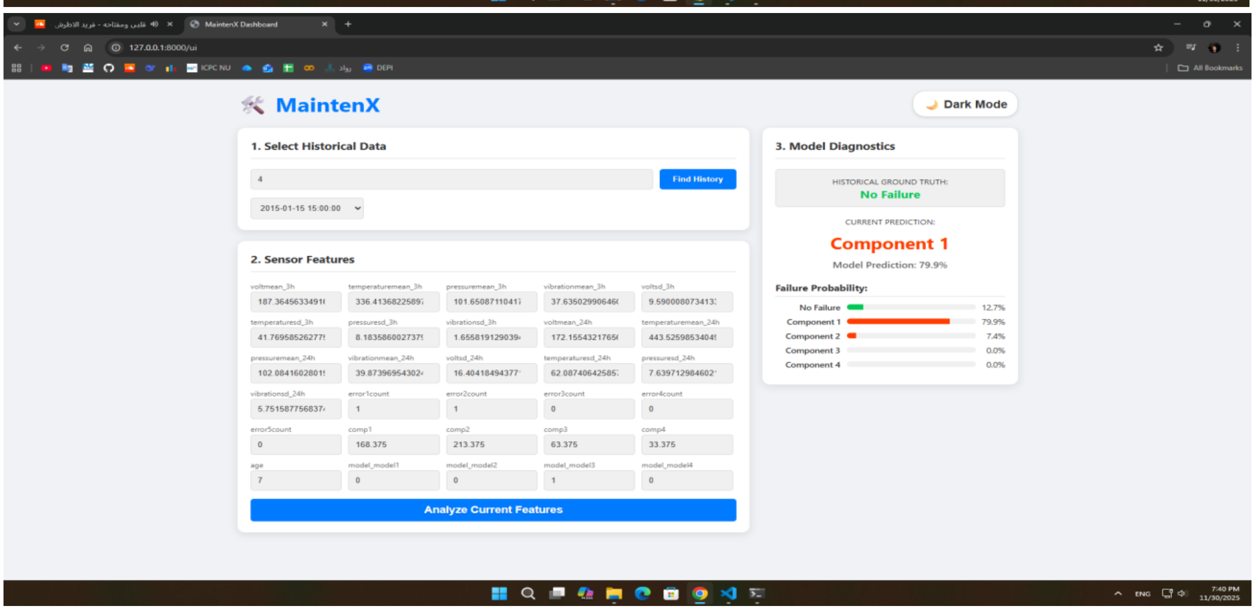
3. **Cost-Sensitive Learning:** Incorporate a "Cost Matrix" into the training, where the cost of a Missed Failure is weighed ($10,000) against the cost of a False Alarm ($500).

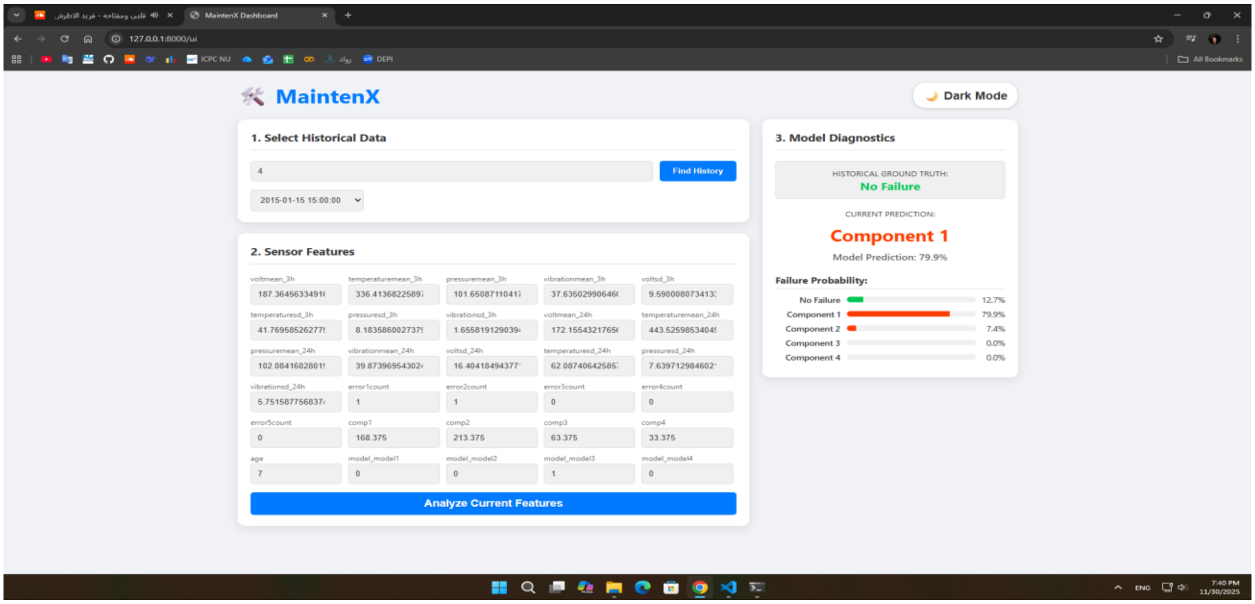
10.3 Resources

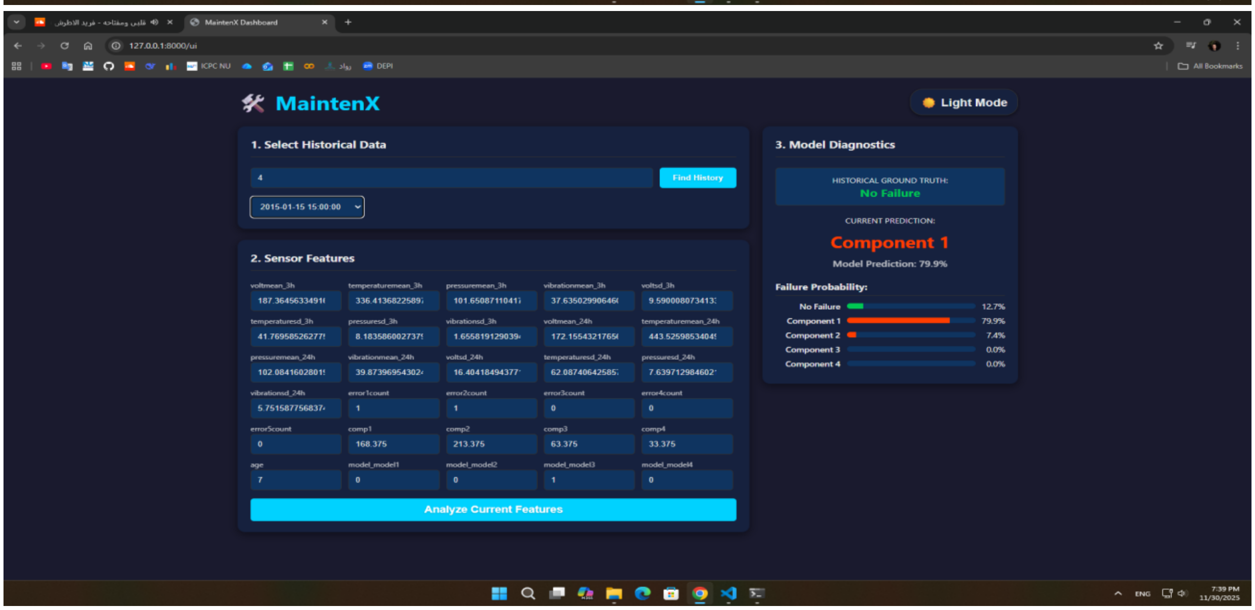
- **GitHub Repository:** https://github.com/MohamedElsadany56/MaintenX/tree/main
- **Live App:** https://maintenx-app-egy.azurewebsites.net/ui

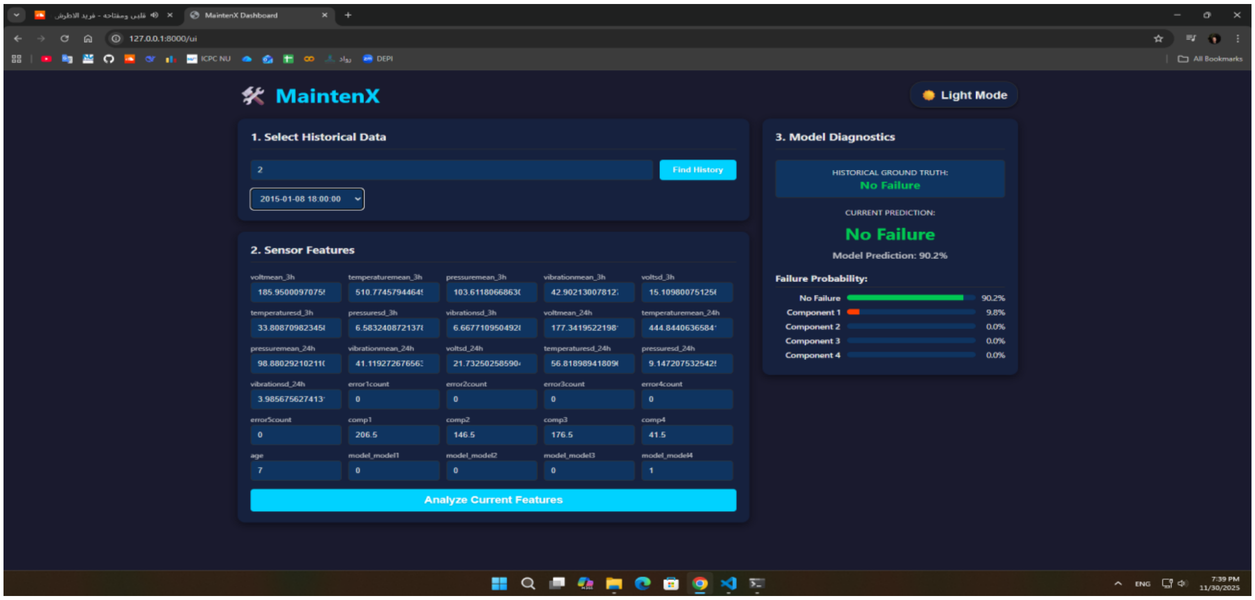
11.1 Screenshots of the UI:

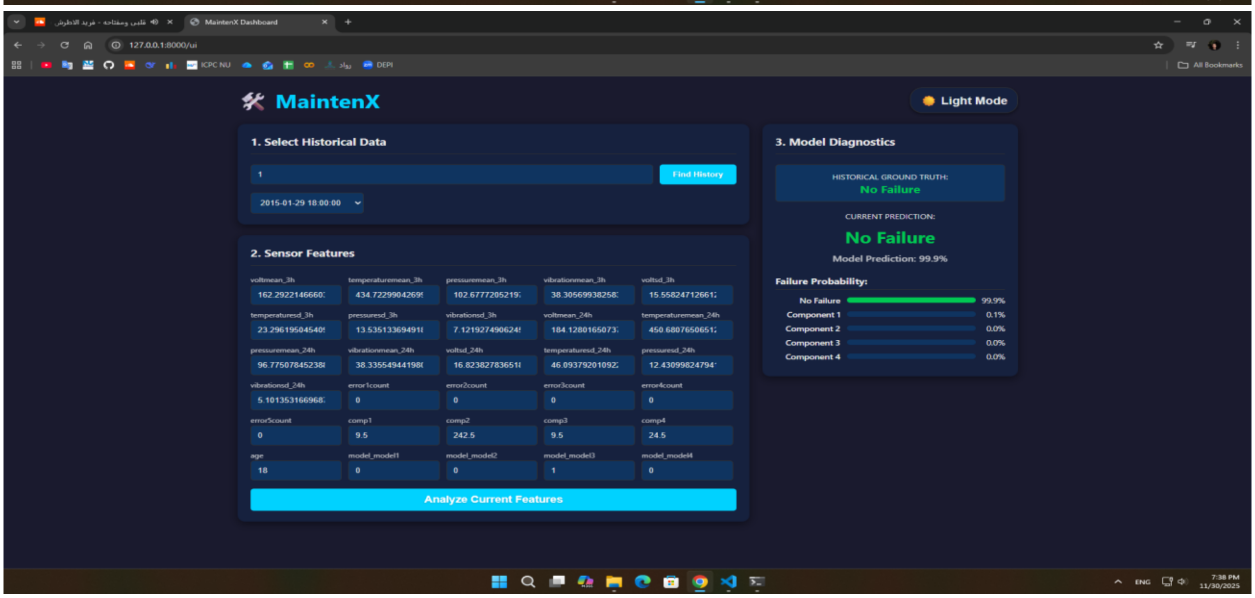




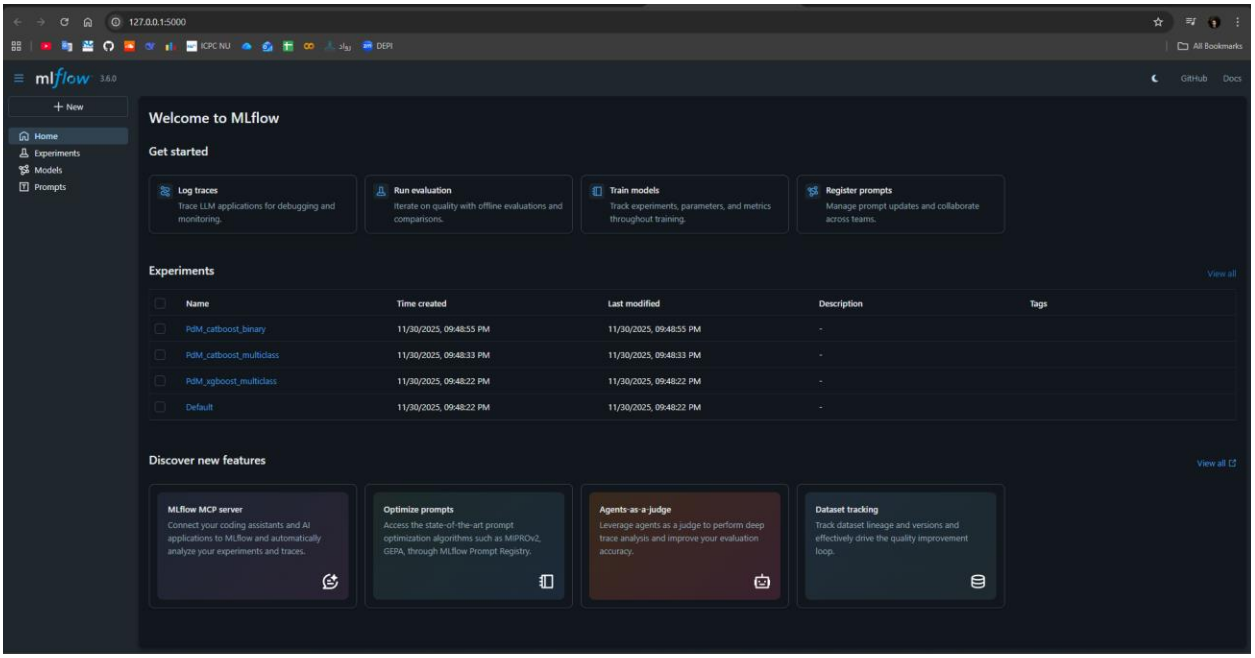

MaintenX Dashboard
127.0.0.1:8000/ui
MaintenX
Dark Mode
1. Select Historical Data
4
Find History
2015-01-15 15:00:00
2. Sensor Features
voltmean_3h
temperaturemean_3h
pressuremean_3h
vibrationmean_3h
voltsd_3h
temperaturesd_3h
pressuresd_3h
vibrationsd_3h
voltmean_24h
temperaturemean_24h
pressuremean_24h
vibrationmean_24h
voltsd_24h
temperaturesd_24h
pressuresd_24h
vibrationsd_24h
error1count
error2count
error3count
error4count
error5count
comp1
comp2
comp3
comp4
age
model_model1
model_model2
model_model3
model_model4
Analyze Current Features
3. Model Diagnostics
HISTORICAL GROUND TRUTH:
No Failure
CURRENT PREDICTION:
Component 1
Model Prediction: 79.9%
Failure Probability:
No Failure 12.7%
Component 1 79.9%
Component 2 7.4%
Component 3 0.0%
Component 4 0.0%
7:40 PM
11/30/2025


MaintenX Dashboard
127.0.0.1:8000/ui
MaintenX
Light Mode
1. Select Historical Data
4
Find History
2015-01-15 15:00:00
2. Sensor Features
Analyze Current Features
3. Model Diagnostics
HISTORICAL GROUND TRUTH:
No Failure
CURRENT PREDICTION:
Component 1
Model Prediction: 79.9%
Failure Probability:
No Failure 12.7%
Component 1 79.9%
Component 2 7.4%
Component 3 0.0%
Component 4 0.0%
7:39 PM
11/30/2025





11.2 MLFlow Experiments:


127.0.0.1:5000
ICPC NU
DEPI
All Bookmarks
mlflow 3.6.0
GitHub
Docs
+ New
Home
Experiments
Models
Prompts
Welcome to MLflow
Get started
Log traces
Trace LLM applications for debugging and monitoring.
Run evaluation
Iterate on quality with offline evaluations and comparisons.
Train models
Track experiments, parameters, and metrics throughout training.
Register prompts
Manage prompt updates and collaborate across teams.
Experiments
View all
Name
Time created
Last modified
Description
Tags
PdM_catboost_binary
11/30/2025, 09:48:55 PM
11/30/2025, 09:48:55 PM
PdM_catboost_multiclass
11/30/2025, 09:48:33 PM
11/30/2025, 09:48:33 PM
PdM_xgboost_multiclass
11/30/2025, 09:48:22 PM
11/30/2025, 09:48:22 PM
Default
11/30/2025, 09:48:22 PM
11/30/2025, 09:48:22 PM
Discover new features
View all
MLflow MCP server
Connect your coding assistants and AI applications to MLflow and automatically analyze your experiments and traces.
Optimize prompts
Access the state-of-the-art prompt optimization algorithms such as MIPROv2, GEPA, through MLflow Prompt Registry.
Agents-as-a-judge
Leverage agents as a judge to perform deep trace analysis and improve your evaluation accuracy.
Dataset tracking
Track dataset lineage and versions and effectively drive the quality improvement loop.