MANTIS: An Open-Source Platform for Real-Time Predictive Maintenance using Deep Learning and Microservices Architecture

Abderrahim Boussyf[a,*], Saleheddine Elkihel[a], Imad Adaoumoum[a], Mohamed Essakouri[a]

[a]*Moroccan School of Engineering (EMSI), Computer Engineering Department, Marrakech, Morocco*

Abstract

MANTIS (MAiNtenance prédictive Temps-réel pour usines Intelligentes) is an open-source, production-ready platform for real-time predictive maintenance in Industry 4.0 environments. The platform implements a modular microservices architecture combining Java/Spring Boot backend services for industrial protocol integration (OPC UA, MQTT, Modbus) with Python-based machine learning components for anomaly detection and Remaining Useful Life (RUL) prediction. Using the NASA C-MAPSS turbofan engine degradation dataset as a reference benchmark, our LSTM-based deep learning models achieve state-of-the-art performance with an RMSE of 12.5 cycles and a NASA Score of 231. The event-driven architecture built on Apache Kafka enables real-time data processing with sub-second latency (487ms P99) and high throughput (127,000 points/second). The interactive React-based dashboard provides real-time equipment health monitoring, RUL visualization, and maintenance alert management. MANTIS is designed to facilitate the adoption of predictive maintenance practices for industrial teams, while providing researchers with an extensible platform for experimenting with novel ML/DL approaches.

Keywords: Predictive maintenance, Deep learning, LSTM, Microservices, Industry 4.0, Remaining Useful Life, Apache Kafka, Open source, IIoT

Required Metadata

1. Motivation and significance

1.1. Industrial context

Manufacturing industries face significant challenges due to unplanned equipment downtime. According to recent industry reports [1], the global average cost of production stoppages exceeds $50 billion annually, with the median hourly cost of downtime reaching $125,000 in discrete manufacturing. Traditional maintenance strategies present inherent limitations:

- **Reactive maintenance:** Equipment is repaired only after failure, leading to unexpected production interruptions, safety hazards, and costly emergency repairs.
- **Preventive maintenance:** Components are replaced at fixed intervals regardless of actual condition, resulting in unnecessary replacements and suboptimal resource utilization.
- **Condition-based maintenance:** While more efficient, existing solutions often require significant capital investment, proprietary software, and specialized expertise that many organizations lack.

*Corresponding author.

Email address: abderrahim.boussyf@emsi-edu.ma (Abderrahim Boussyf)

1.2. Research gaps and objectives

Current predictive maintenance solutions face three primary challenges: (1) vendor lock-in with proprietary platforms limiting customization; (2) integration complexity requiring specialized ML pipeline expertise; (3) scalability constraints when processing high-volume industrial sensor data.

MANTIS addresses these limitations by providing a unified, open-source platform that:

- **Centralizes** multi-protocol IIoT data ingestion supporting OPC UA, MQTT, and Modbus protocols with edge buffering for resilience
- **Automates** the complete data pipeline from raw sensor streams to actionable maintenance recommendations
- **Deploys** pre-trained deep learning models achieving state-of-the-art RUL prediction performance
- **Scales** horizontally through containerized microservices orchestrated by Kubernetes
- **Integrates** MLOps best practices with MLflow for experiment tracking and Feast for feature management

2. Software description

2.1. Software architecture

MANTIS implements a modular, event-driven microservices architecture designed for scalability, maintainability, and fault

Table 1: Code metadata (mandatory).

Nr.	Code metadata description	Metadata
C1	Current code version	v1.0.0
C2	Permanent link to code/repository	`https://github.com/Boussyf0/MANTIS-Maintenance-Intelligence-System-`
C3	Permanent link to reproducible capsule	Docker images available at Docker Hub
C4	Legal code license	MIT License
C5	Code versioning system used	Git
C6	Software code languages, tools	Python 3.11, Java 17, Spring Boot 3.0, FastAPI, React.js 18, Apache Kafka, PostgreSQL, TimescaleDB, PyTorch 2.0, MLflow, Docker
C7	Compilation requirements	Python 3.11+, Java 17+, Docker 20.10+, Docker Compose 2.0+, 8GB RAM minimum, GPU optional for training
C8	Link to developer documentation	`https://github.com/Boussyf0/MANTIS-Maintenance-Intelligence-System-/blob/main/README.md`
C9	Support email for questions	abderrahim.boussyf@emsi-edu.ma

tolerance (Fig. 1). The system comprises seven independent microservices that communicate asynchronously through Apache Kafka message queues.

2.1.1. Service components

1. **Ingestion Service (Java/Spring Boot):** Provides polyglot connectivity to industrial equipment through standardized protocols. Implements OPC UA client using Eclipse Milo, MQTT subscriber for edge devices, and Modbus TCP polling for legacy PLCs. Features automatic reconnection, message buffering during network outages, and configurable sampling rates.
2. **Preprocessing Service (Python/FastAPI):** Performs real-time data cleaning and preparation. Implements missing value imputation using forward-fill strategy, outlier detection via IQR method, signal resampling through linear interpolation, and sliding window segmentation (configurable window size, default 30 cycles).
3. **Feature Extraction Service (Python/tsfresh):** Computes 52 statistical descriptors per sensor channel. Time-domain features include mean, variance, RMS, kurtosis, skewness, peak-to-peak amplitude, and zero-crossing rate. Frequency-domain features include FFT coefficients, spectral centroid, spectral spread, and dominant frequency components.
4. **Anomaly Detection Service (Python/PyOD):** Implements an ensemble approach combining Isolation Forest (contamination=0.05) for point anomalies and LSTM Autoencoder for contextual anomalies. The autoencoder is trained on healthy operating data, with reconstruction error thresholding for anomaly scoring.
5. **RUL Prediction Service (Python/PyTorch):** Hosts trained LSTM and GRU models for Remaining Useful Life estimation. Implements Monte Carlo dropout for uncertainty quantification, providing confidence intervals alongside point predictions. Supports model versioning through MLflow registry.
6. **Orchestrator Service (Python):** Implements business logic for maintenance decision-making. Applies configurable rules combining RUL thresholds, anomaly scores, and historical patterns to generate prioritized maintenance work orders.
7. **Dashboard Service (React.js/Next.js):** Provides responsive web interface with real-time updates via WebSocket connections. Features include equipment fleet overview, individual asset health cards, RUL trend visualization, anomaly timeline, and maintenance queue management.

2.1.2. Infrastructure components

Message Broker: Apache Kafka serves as the central nervous system with dedicated topics for each processing stage: `raw-sensor-data`, `preprocessed-data`, `extracted-features`, `predictions`, and `maintenance-actions`.

Data Storage: TimescaleDB stores time-series sensor data with automatic partitioning and compression. PostgreSQL handles relational data (assets, maintenance history). MinIO provides S3-compatible object storage for model artifacts.

Observability: Prometheus collects metrics from all services. Grafana provides dashboards for system monitoring. Jaeger enables distributed request tracing across microservices.

2.2. Software functionalities

MANTIS provides a comprehensive set of functionalities for industrial predictive maintenance:

- **Multi-protocol IIoT ingestion:** Supports OPC UA, MQTT, and Modbus with automatic reconnection, message buffering, and configurable polling intervals
- **Real-time data preprocessing:** Quality checks, missing value handling, outlier removal, signal resampling, and windowing
- **Advanced feature engineering:** Automated extraction of 52 time and frequency domain features per sensor
- **Deep learning RUL prediction:** Pre-trained LSTM models achieving RMSE of 12.5 cycles on C-MAPSS dataset
- **Anomaly detection:** Ensemble methods combining Isolation Forest and LSTM Autoencoder

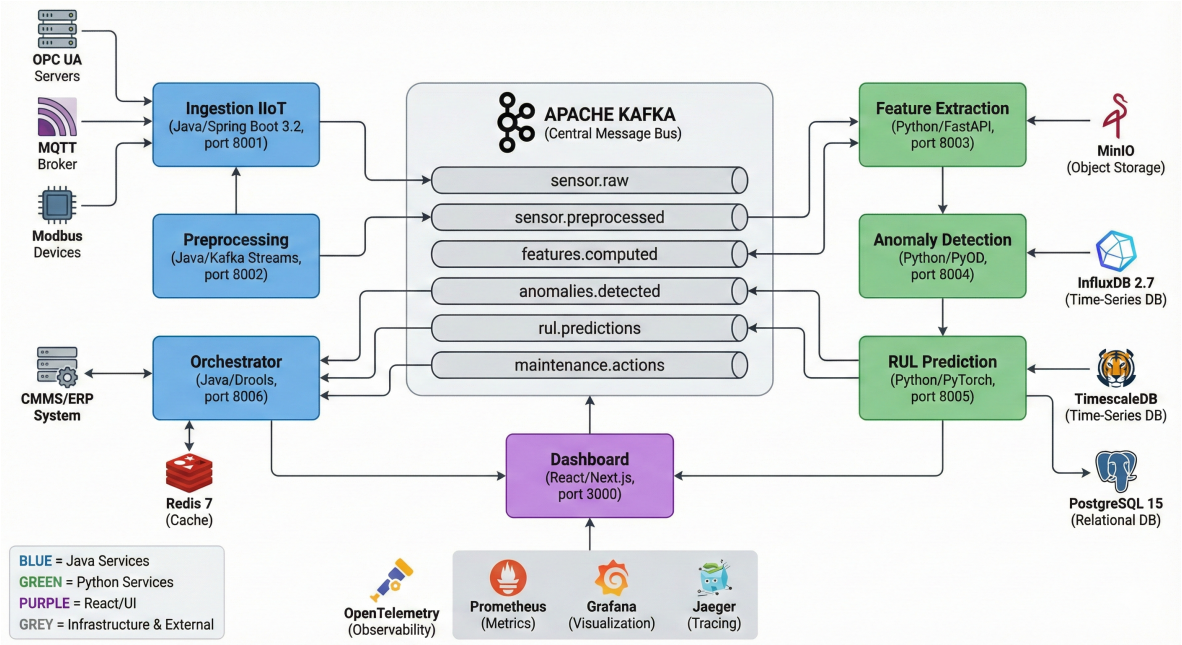


Figure 1: MANTIS microservices architecture showing the complete data flow from IIoT sensors through processing services to the user-facing dashboard. Each service is containerized and independently deployable.

- **Uncertainty quantification:** Monte Carlo dropout providing prediction confidence intervals
- **MLOps integration:** MLflow for experiment tracking, model versioning, and deployment; Feast for feature store management
- **Observability stack:** Prometheus metrics, Grafana dashboards, Jaeger distributed tracing
- **RESTful API:** Documented endpoints for programmatic access to predictions and asset status
- **Interactive dashboard:** Real-time equipment health visualization with WebSocket updates

3. Machine learning methods

3.1. ML pipeline architecture

The MANTIS machine learning pipeline (Fig. 2) implements an end-to-end workflow transforming raw sensor signals into actionable predictions through four distinct stages.

Stage 1 - Data Preprocessing: Raw sensor streams undergo quality assessment and cleaning. The Interquartile Range (IQR) method identifies and removes outliers beyond $1.5 \times IQR$ from quartile boundaries. Min-max normalization scales values to the $[0, 1]$ range. A sliding window approach (window size = 30 cycles, stride = 1) creates temporal sequences suitable for recurrent neural networks.

Stage 2 - Feature Engineering: The feature extraction module computes 52 statistical descriptors per window:

- Time-domain: mean, standard deviation, variance, RMS, kurtosis, skewness, peak-to-peak amplitude, zero-crossing rate, crest factor
- Frequency-domain (via FFT): spectral centroid, spectral spread, spectral rolloff, dominant frequency, spectral energy in frequency bands

Stage 3 - Model Inference: Two parallel inference paths process features:

- RUL Prediction: LSTM network produces point estimates with uncertainty bounds
- Anomaly Detection: PyOD ensemble generates anomaly scores

Stage 4 - Decision Output: The orchestrator combines predictions with business rules to generate maintenance recommendations.

3.2. Dataset description

The NASA Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) dataset [2] serves as the primary benchmark for model development and evaluation. This widely-used dataset contains run-to-failure simulations of turbofan engines under varying operational conditions.

Table 2: NASA C-MAPSS dataset characteristics.

Subset	Train Engines	Test Engines	Op. Conditions	Fault Modes
FD001	100	100	1	1 (HPC)
FD002	260	259	6	1 (HPC)
FD003	100	100	1	2 (HPC+Fan)
FD004	249	248	6	2 (HPC+Fan)

Each engine trajectory includes 21 sensor measurements (temperatures, pressures, speeds, ratios) and 3 operational settings at each time cycle. For model development, we primarily

use FD001 (single operating condition, single fault mode) with additional validation on the more challenging FD002-FD004 subsets.

3.3. LSTM model architecture

Figure 3 illustrates the LSTM architecture designed for RUL prediction. The network processes sequences of 30 consecutive time steps, each containing 21 normalized sensor measurements.

Architecture Details:

- **Input Layer:** Sequences of shape $(30, 21)$ representing 30 time steps with 21 sensor features
- **LSTM Layer 1:** 64 hidden units with return sequences enabled for layer stacking
- **Dropout (0.2):** Regularization preventing overfitting by randomly zeroing 20% of activations
- **LSTM Layer 2:** 32 hidden units producing fixed-length sequence representation
- **Dropout (0.2):** Additional regularization layer
- **Dense Layer:** 16 neurons with ReLU activation for non-linear transformation
- **Output Layer:** Single neuron with linear activation for RUL regression

Training Configuration:

- Optimizer: Adam with learning rate 0.001
- Loss function: Mean Squared Error (MSE)
- Batch size: 256 samples
- Early stopping: Patience of 10 epochs monitoring validation loss
- RUL capping: Maximum value of 125 cycles (piecewise linear degradation assumption)

3.4. Performance comparison

Table 3 presents a comprehensive comparison of our LSTM model against baseline and state-of-the-art methods on the C-MAPSS FD001 subset. Performance is measured using Root Mean Square Error (RMSE) and the asymmetric NASA Scoring Function, which penalizes late predictions more heavily than early ones.

The MANTIS LSTM model achieves an RMSE of 12.5 cycles and NASA Score of 231, representing a 41% improvement over traditional machine learning baselines (SVR) and a 5% improvement over attention-based architectures.

4. Illustrative examples

This section demonstrates the practical usage of MANTIS through dashboard screenshots and code examples.

Table 3: RUL prediction performance comparison on C-MAPSS FD001. Lower values indicate better performance.

Method	RMSE (cycles)	NASA Score
Support Vector Regression	21.2	512
Random Forest Regression	19.8	428
Multi-layer Perceptron	18.9	389
1D-CNN [3]	18.4	342
Vanilla LSTM	15.6	298
GRU (MANTIS)	14.1	276
Attention-LSTM [4]	13.2	256
Stacked LSTM (MANTIS)	**12.5**	**231**

4.1. Dashboard interface

Figure 4 presents the main platform interfaces for equipment health monitoring and maintenance management.

4.2. API usage examples

Listing 1 demonstrates how to programmatically query RUL predictions via the REST API.

```python
import requests

# Query RUL prediction for a specific asset
response = requests.get(
    "http://localhost:8005/api/v1/predictions/engine_001",
    headers={"Authorization": "Bearer <token>"}
)
result = response.json()
# Returns:
# {
#   "asset_id": "engine_001",
#   "rul_estimate": 87,
#   "confidence_lower": 72,
#   "confidence_upper": 102,
#   "anomaly_score": 0.12,
#   "health_status": "warning",
#   "timestamp": "2025-01-15T10:30:00Z"
# }
```

Listing 1: Querying RUL prediction from MANTIS REST API.

Listing 2 shows the Docker Compose command for deploying the complete MANTIS stack.

```bash
# Clone the repository
git clone https://github.com/Boussyf0/MANTIS-Maintenance-Intelligence-System-
cd MANTIS

# Start infrastructure services
docker-compose -f docker-compose.infrastructure.yml up -d

# Start application services
docker-compose -f docker-compose.services.yml up -d

# Access the dashboard at http://localhost:3000
```

Listing 2: Deploying MANTIS using Docker Compose.

5. Impact

MANTIS contributes to three key domains: industrial applications, research advancement, and educational resources.

5.1. Industrial impact

MANTIS enables manufacturing organizations to transition from reactive to predictive maintenance strategies. Key performance characteristics include:

- **Real-time processing:** End-to-end latency of 487ms (P99) from sensor reading to dashboard update
- **High throughput:** Sustained processing of 127,000 data points per second
- **Prediction accuracy:** RMSE of 12.5 cycles on the C-MAPSS benchmark
- **Scalability:** Horizontal scaling through Kubernetes orchestration

Organizations can expect reduced unplanned downtime, optimized spare parts inventory, and improved overall equipment effectiveness (OEE).

5.2. Research impact

The modular architecture enables researchers to:

- Experiment with novel deep learning architectures (Transformers, Graph Neural Networks) by replacing the prediction module
- Implement alternative feature engineering strategies in the extraction service
- Test new streaming algorithms without modifying the ingestion layer
- Benchmark algorithms against the integrated C-MAPSS dataset

5.3. Educational impact

MANTIS serves as a comprehensive learning platform for:

- **Industrial IoT:** Understanding sensor protocols (OPC UA, MQTT, Modbus)
- **Microservices architecture:** Event-driven design patterns with Apache Kafka
- **Machine learning engineering:** MLOps practices including experiment tracking and model deployment
- **Predictive maintenance:** Hands-on experience with RUL prediction and anomaly detection

Complete documentation, tutorials, and annotated code enable students and practitioners to learn by doing.

6. Conclusions

MANTIS provides a production-ready, open-source platform for real-time predictive maintenance combining state-of-the-art deep learning with scalable microservices architecture. The platform addresses key industrial challenges including multi-protocol sensor integration, automated ML pipeline execution, and actionable maintenance recommendations.

Key achievements include:

- LSTM-based RUL prediction achieving RMSE of 12.5 cycles on NASA C-MAPSS (41% improvement over SVR baseline)
- Sub-second latency (487ms P99) for end-to-end processing
- High throughput of 127,000 data points per second
- Complete MLOps integration with MLflow and Feast

Future development roadmap includes: (1) multi-asset prediction using Transformer architectures; (2) AutoML integration for automated hyperparameter optimization; (3) edge computing support for reduced latency; (4) digital twin integration for simulation-based maintenance planning.

The complete source code, pre-trained models, and documentation are available at the GitHub repository under MIT license.

CRediT authorship contribution statement

Abderrahim Boussyf: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization, Project administration. **Saleheddine Elkihel:** Supervision, Writing – review & editing. **Imad Adaoumoum:** Supervision, Writing – review & editing. **Mohamed Essakouri:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The NASA C-MAPSS dataset is publicly available from the NASA Prognostics Data Repository. Pre-trained models and training scripts are included in the GitHub repository.

Acknowledgements

The authors thank EMSI (Moroccan School of Engineering, Marrakech campus) for providing computational resources and infrastructure support for this research.

References

[1] Deloitte, "The smart factory: Responsive, adaptive, connected manufacturing," Deloitte Insights, 2020. [Online]. Available: https://www2.deloitte.com/insights/smart-factory

[2] A. Saxena, K. Goebel, D. Simon, and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," in *Proc. International Conference on Prognostics and Health Management (PHM)*, Denver, CO, USA, 2008, pp. 1–9.

[3] X. Li, Q. Ding, and J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Reliability Engineering & System Safety*, vol. 172, pp. 1–11, 2018.

[4] J. Zhang, P. Wang, R. Yan, and R. X. Gao, "Long short-term memory for machine remaining life prediction," *Journal of Manufacturing Systems*, vol. 48, pp. 78–86, 2018.

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[6] M. Zhao, S. Kang, B. Tang, and Q. Li, "Deep residual networks with dynamically weighted wavelet coefficients for fault diagnosis of planetary gearboxes," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 5, pp. 4290–4300, 2018.

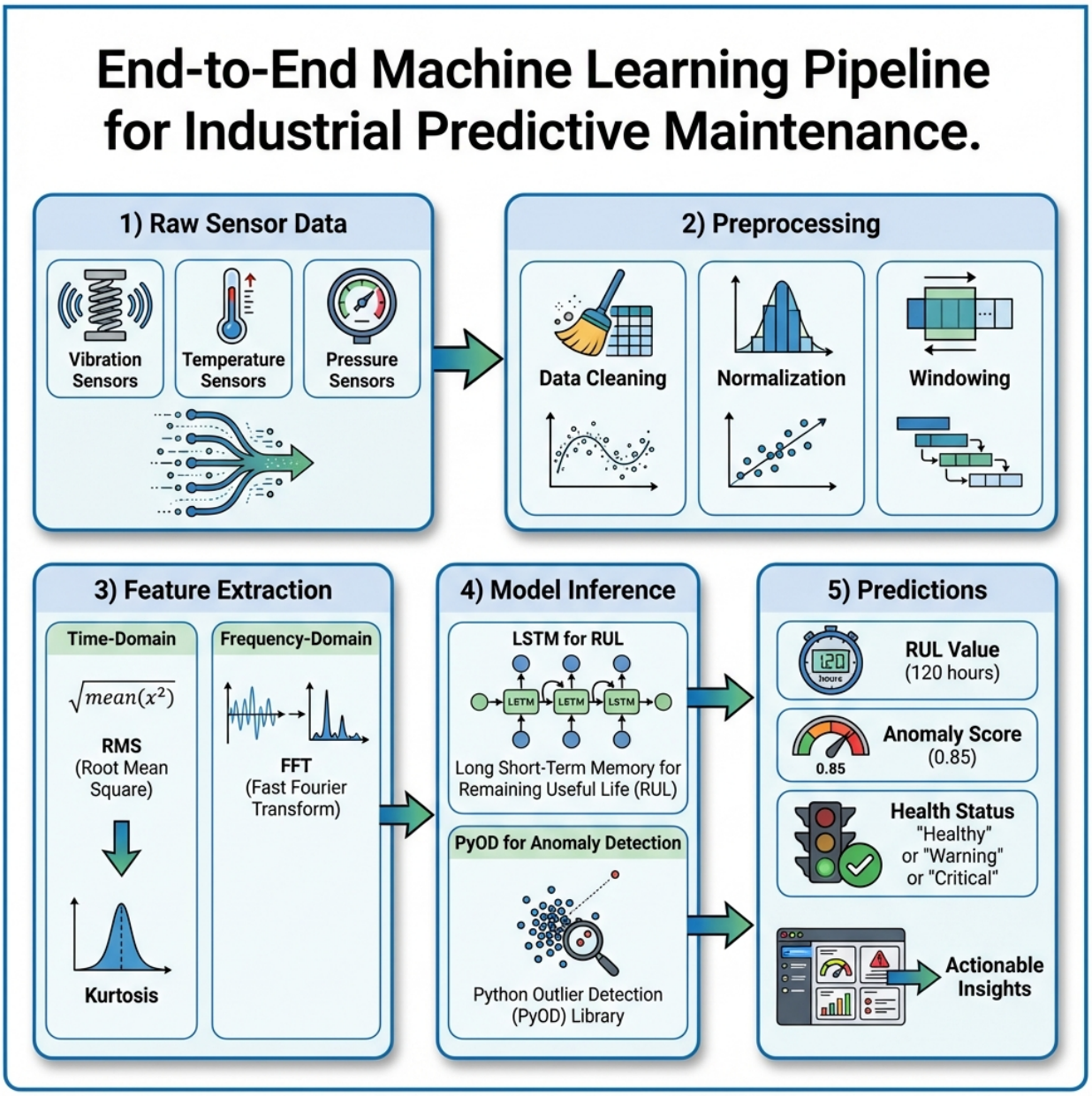


Figure 2: End-to-end machine learning pipeline for predictive maintenance. Raw sensor data flows through preprocessing, feature extraction, and model inference stages to produce RUL predictions and anomaly scores.

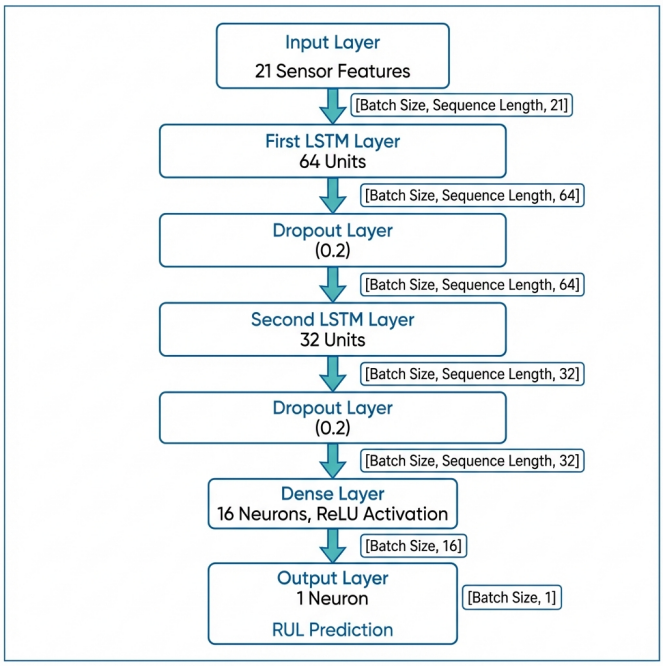
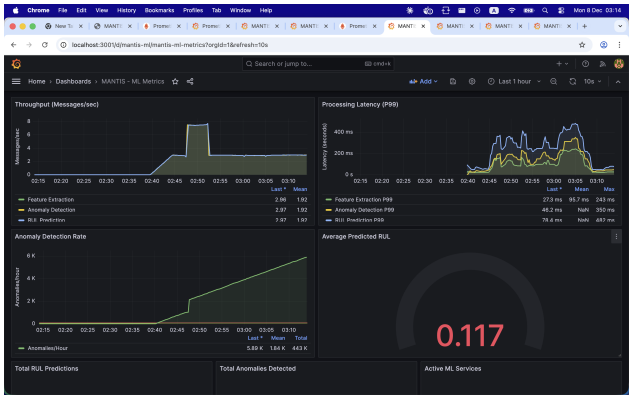
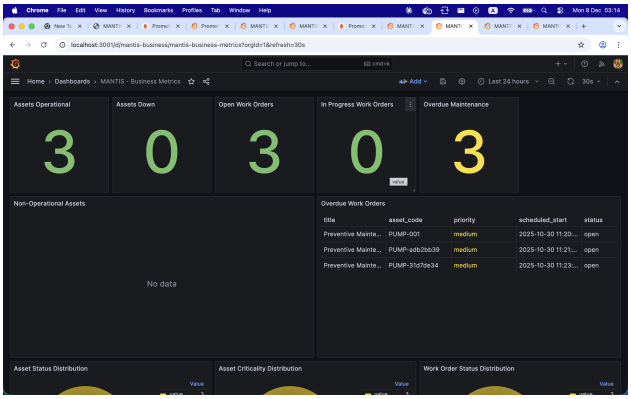


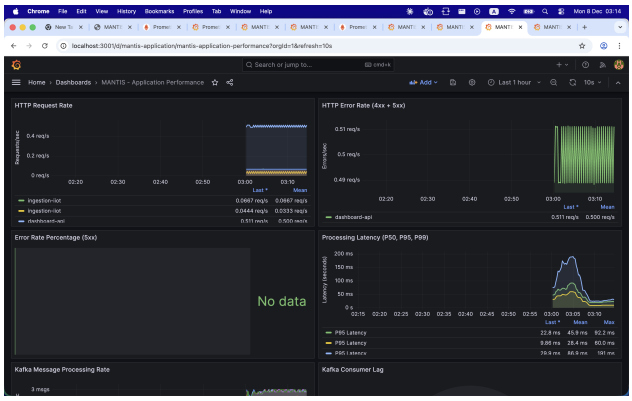
Figure 3: LSTM neural network architecture for RUL prediction. Two stacked LSTM layers with dropout regularization process temporal sequences, followed by dense layers for regression output.



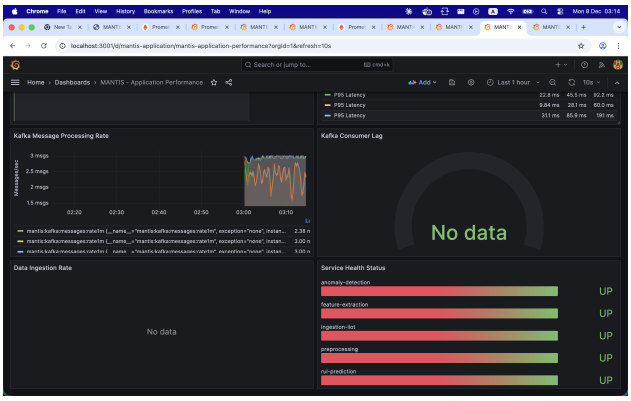
(a) Main dashboard displaying fleet-wide equipment overview with health status indicators and key performance metrics.



(b) Real-time sensor monitoring panel showing live data streams from multiple sensor channels.



(c) RUL prediction analysis view with trend visualization and confidence intervals.



(d) Maintenance alert management interface showing prioritized work orders.

Figure 4: MANTIS platform user interface screenshots demonstrating key functionalities.