Final Project Report: Interpretable Machine Learning for Industrial Equipment Health Monitoring

Samuel Adetsi, Mu Ha, Cheng Zhang, Michael Hewlett

2025-06-01

Table of contents

1 Executive Summary

Brilliant Automation's existing system for equipment health rating relied on opaque MATLAB heuristics that lacked interpretability and automation. To address these limitations, we designed a transparent, end-to-end machine learning pipeline deployed on AWS. High-frequency vibration and temperature data (5-second intervals) from tube mills, belt conveyors, and high-temperature fans are ingested, logged, and synchronized with 20-minute human-annotated health ratings. Robust preprocessing merges, pivots, and window-aggregates sensor streams, while feature engineering scripts extract spectral DSP metrics and time-domain summaries to yield a rich feature set for modeling.

We evaluated seven algorithm classes—from a mean-based baseline through linear (Ridge, Polynomial Ridge) and kernel (SVR) models to ensemble trees (Random Forest, XGBoost) and a simple RuleTree—training each separately on 12–15 health rating targets. Time-series cross-validation (5 splits) and two core metrics (RMSE, R^2) guided model selection. Ensemble trees dominated most targets, with Random Forest achieving R^2 up to 0.471 on velocity RMS and XGBoost excelling on frequency-domain ratings. SVR provided competitive performance for mechanical fit assessments, demonstrating the importance of model diversity.

The final system delivers a real-time dashboard and AWS-automated data pipeline, reducing manual effort and improving transparency for maintenance engineers. Short-term recommendations include automated CI/CD and serverless alerts; longer-term plans focus on data expansion and anomaly detection to strengthen model robustness. This solution lays a scalable, explainable foundation for predictive maintenance across diverse industrial settings.

2 Introduction

2.1 Problem Context

Brilliant Automation, a leader in industrial monitoring systems based in Shanghai, supports predictive maintenance for critical equipment at a limestone processing facility. Their existing system uses proprietary MATLAB algorithms to calculate health ratings for machines based on high-frequency vibration and temperature sensor data. However, clients have expressed concerns over the opacity of these algorithms and the lack of interpretability behind rating decisions.

To address this, Brilliant Automation partnered with our team to develop a transparent machine learning pipeline capable of matching the performance of the current system while enabling greater insight into the rationale behind predictions.

The project focuses on three types of industrial equipment: a tube mill, a belt conveyor (#8), and a high-temperature fan (#1). Each device is outfitted with sensors measuring four physical signals (acceleration, vibration velocity, temperature) at various locations and intervals.

2.2 Data Description

The dataset included multi-frequency sensor readings collected at 5-second intervals from 15 monitoring points across three major equipment types: tube mills, conveyor belts, and high-temperature fans. Each sensor location captured four key physical quantities: high-frequency acceleration, low-frequency acceleration, vibration velocity, and temperature. Ground truth health ratings were recorded every 20 minutes by maintenance staff across 12–15 categories depending on the device (e.g., alignment status, bearing lubrication, rotor balance, fit condition, crest factor, kurtosis, etc.). This resulted in a temporal misalignment between high-resolution sensor data and sparse rating annotations, requiring careful feature aggregation and validation.

2.3 Objectives

To address this, we defined the following tangible objectives:

- Develop interpretable models to predict 12–15 equipment health rating, based on the device
- Build a dashboard for real-time equipment monitoring
- Automate data ingestion, transformation, and model deployment

These objectives align with the client's priorities: explainability, accuracy, and scalability.

3 Data Science Methods

3.1 Data Preprocessing

We began by establishing a robust logging framework to trace every step of data ingestion and transformation. A dedicated `preprocessing.log` file captures INFO-level messages that include file discovery events, metadata summaries, and error conditions. This ensures reproducibility and simplifies debugging when processing large volumes of time-series data.

Sensor readings and health ratings are loaded via the `read_device_files()` function, which supports both local directories and AWS S3 buckets. Files are matched using regular expressions on device names, and separate sheets for different sensor locations or rating metrics are concatenated into unified DataFrames. This approach accommodates new devices or sensor points without altering core logic.

Once raw tables are loaded, we merge `Date` and `Time` columns to create a `datetime` index, drop redundant columns, and pivot long-form measurements into a wide format where each sensor measurement becomes its own column. Forward-filling handles intermittent missing temperature readings, while as-of merging aligns each high-frequency sensor timestamp with the next available 20-minute health rating. The final merged dataset, containing synchronized sensor and rating values, is exported as a CSV or uploaded to S3 for downstream analysis.

While forward-filling aligns individual sensor timestamps to the next rating, it often results in many duplicated values in a dataset where ratings exhibit limited variance and incurs unnecessary computation on identical annotations. Instead, we implement a bucket summarization approach during

feature engineering. Aggregating measurements into consistent 20-minute windows reduces redundant rating duplication, captures true temporal patterns more effectively, and is computationally cheaper than propagating the same rating across high-frequency timestamps.

3.2 Feature Engineering and Selection

After preprocessing, the `feature_eng.py` script enhances the dataset with frequency-domain insights. It scans every JSON file under `Data/voltage/`, extracts `axisX` and `axisY` data to compute sampling frequency, and applies digital signal processing (DSP) functions to derive metrics like velocity RMS, crest factor, kurtosis, and band-specific RMS and peak values. These metrics capture subtle anomalies in vibration signals that are not apparent in time-domain summaries.

Next, the script reads the merged sensor-rating CSV, appends `_rating` to each health rating column, and groups data into 20-minute buckets by location. For each bucket, we compute count, mean, standard deviation, minimum, and maximum for core measurements. This bucketing respects rating changes and time-based windowing, ensuring that each aggregated feature set corresponds to a consistent health annotation.

Finally, DSP metrics and bucketed summaries are joined using interval-index lookups, resulting in a comprehensive feature table. This table includes synchronized time-domain statistics and spectral descriptors, providing rich inputs for model training without manual feature crafting for each new device type.

3.3 Modeling Strategy

The modeling pipeline loads the full feature set (`*_full_features.csv`), drops extraneous identifiers, and discards any rows with missing values. We convert `datetime` into a numeric timestamp, one-hot encode categorical variables (`location` and `wave_code`), and standardize numeric features. For polynomial Ridge models, degree-2 interaction terms are generated to capture simple non-linear relationships.

We prototyped eight model architectures—including a recurrent neural network (RNN)—but selected seven for detailed analysis due to data limitations and interpretability requirements: a baseline `DummyRegressor`, linear Ridge Regression, Polynomial Ridge (degree 2), Random Forest, XGBoost, Support Vector Regression (SVR), and a one-level RuleTree. When the `--tune` flag is enabled, each model undergoes `RandomizedSearchCV` with time-series splitting to optimize hyperparameters over multiple folds.

We evaluate each model separately for every health rating target, resulting in 12–15 distinct training runs per model type. This per-target approach allows each model to specialize in the patterns of individual ratings, although it increases overall computational load and storage requirements compared to a multi-output strategy.

3.3.1 Model Rationale and Interpretability vs. Accuracy

To comprehensively address the partner's needs and choose an appropriate model, we span a spectrum from simple to complex:

- **Baseline (Mean Predictor):** The `DummyRegressor` uses global mean ratings as a naive benchmark. It establishes a performance floor, ensuring more complex models provide meaningful gains.
- **Linear Models (Ridge, Polynomial Ridge):** These line-fitting approaches offer transparency through simple coefficients, enabling stakeholders to understand feature impacts. Polynomial Ridge extends linear fits to capture quadratic patterns but remains relatively interpretable.
- **Tree-Based Models (Random Forest, XGBoost, RuleTree):** Decision trees inherently segment data by thresholds, providing conditional rules that map well to industrial fault detection. Random Forest and XGBoost ensemble trees to boost accuracy, trading off some interpretability for non-linear interaction capture. The shallow RuleTree retains high transparency with limited complexity.
- **Support Vector Regression (SVR):** SVR leverages kernel functions to model complex relationships. While accurate, it is less transparent, making it suitable when accuracy needs outweigh interpretability concerns.

This diverse set allows us to compare predictive performance (e.g., RMSE, R^2) against explainability requirements. Maintenance teams can select models that balance model fidelity with the ability to justify predictions in operational contexts.

3.4 Evaluation Metrics

We measure performance using Root Mean Squared Error (RMSE) and Coefficient of Determination (R^2). RMSE, expressed in rating units, highlights average deviations between predicted and observed values, with a strong focus on penalizing large errors that could delay critical maintenance actions.

R^2 quantifies the proportion of variance in the ground truth ratings explained by the model. Higher R^2 values indicate that the model captures more of the underlying behavior of each health metric, giving engineers confidence in its reliability across operational cycles.

By combining RMSE and R^2, we address both the magnitude and consistency of predictions. These metrics align with stakeholder needs: maintenance teams require clear error tolerances (via RMSE) and an overall reliability score (via R^2) to justify model deployment and to plan intervention strategies.

4 Data Product and Results

Table 1: Best performing model for each health rating target

Target	Best Model	RMSE	R^2
alignment_status_rating	Random Forest	2.997	0.181
rotor_balance_status_rating	Random Forest	2.165	0.115
rubbing_condition_rating	Random Forest	0.534	–0.473
velocity_rms_rating	Random Forest	1.560	0.471

Target	Best Model	RMSE	R^2
bearing_lubrication_rating	XGBoost	0.243	0.154
crest_factor_rating	XGBoost	0.714	–0.325
kurtosis_opt_rating	XGBoost	0.662	0.004
rms_1_10khz_rating	XGBoost	0.066	0.328
peak_value_opt_rating	XGBoost	0.603	–0.068
fit_condition_rating	SVR	1.769	0.360

Table 1 ensemble tree methods such as Random Forest and XGBoost dominate performance across most health metrics. Their superior accuracy stems from their ability to model non-linear interactions between spectral and time-domain features, automatically select important predictors, and remain robust to outliers and noise in the high-frequency vibration data. Random Forest excels on ratings with complex threshold behaviors (e.g., `velocity_rms_rating`), while XGBoost's gradient boosting framework fine-tunes residual errors to capture subtle patterns in less diverse metrics like `crest_factor_rating` and `kurtosis_opt_rating`. In contrast, linear and kernel models (Ridge, SVR) offer interpretability but cannot fully exploit feature interactions, making them better suited for simpler relationships (e.g., `fit_condition_rating`). This trade-off underscores why tree-based ensembles provide the best balance of flexibility and reliability for industrial equipment health prediction.

Hypotheses for Ensemble Superiority:

- **MATLAB-like threshold logic:** The proprietary MATLAB algorithms likely applied fixed threshold rules to sensor metrics, triggering health rating jumps when certain limits were exceeded. Decision trees naturally encode these cut-points, making ensemble methods well-suited to approximate and improve upon the legacy system.
- **Non-linear feature interactions:** Equipment conditions often arise from combinations of signals (e.g., high vibration coupled with elevated temperature). Random Forest and XGBoost automatically learn these interactions, avoiding the need for handcrafted composite features.
- **Robustness to noise and outliers:** High-frequency sensors produce noisy streams with occasional spikes. Random Forest's averaging mechanism smooths outlier effects, while XGBoost iteratively focuses on genuine residual patterns, improving signal-to-noise fidelity.
- **Implicit feature selection:** With dozens of frequency-band metrics available, many are redundant or irrelevant for certain ratings. Tree-based algorithms inherently prioritize important features through split selection, reducing dimensionality and focusing model capacity on impactful predictors.

4.1 Dashboard

The dashboard contains real-time equipment rating monitoring, alerts, and visualizations.

4.2 AWS Data Pipeline

Our end-to-end data pipeline leverages AWS cloud services to automate the collection, processing, and delivery of equipment health insights. The architecture spans from on-premises data sources to cloud-based analytics, ensuring scalable and reliable operations for Brilliant Automation's predictive maintenance system.

4.2.1 Architecture Overview

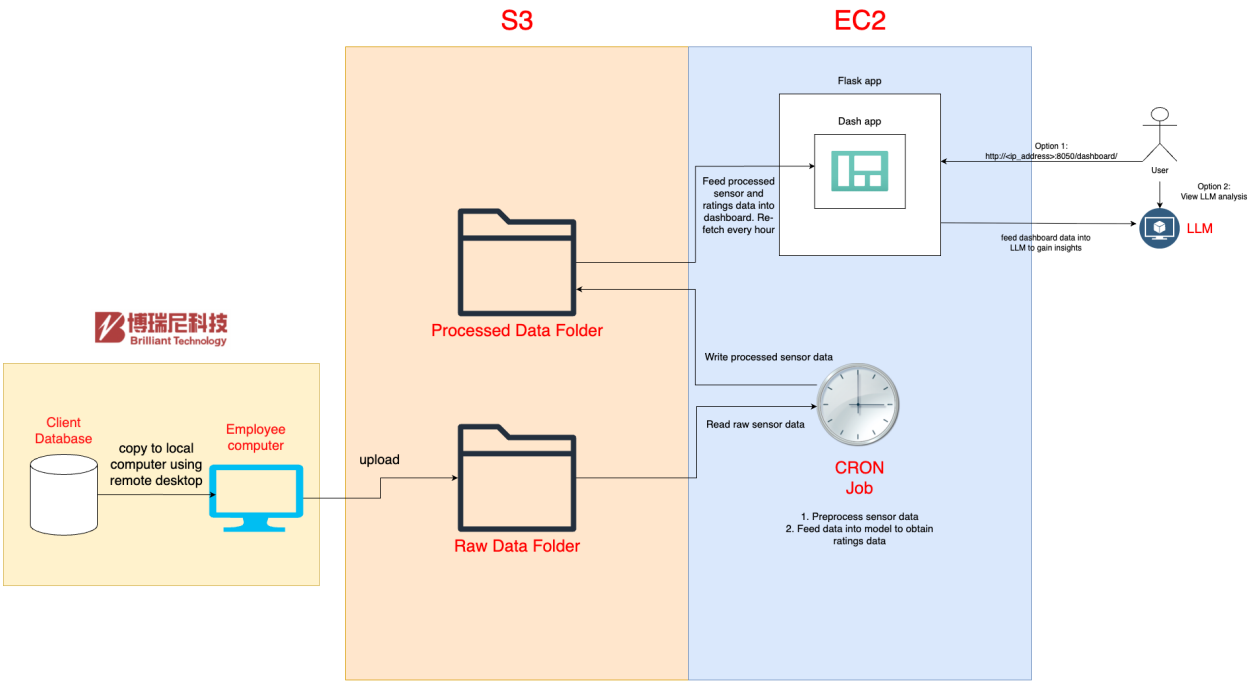

Figure 1: AWS Data Pipeline Architecture

The pipeline follows a three-tier architecture connecting client operations to cloud analytics:

- **Data Collection Layer:** Raw sensor data originates from Brilliant Automation's client database and is transferred to employee computers using remote desktop connections. This maintains security while enabling data access from industrial environments.
- **Storage Layer:** AWS S3 serves as the centralized data repository, hosting both raw sensor files and processed feature datasets. This design enables easy access and sharing between different pipeline components while providing robust backup and versioning capabilities.
- **Processing Layer:** AWS EC2 instances host the data processing scripts, machine learning models, and the interactive dashboard. A dedicated EC2 environment ensures consistent computational resources and simplified dependency management.

4.2.2 Key AWS Services

Amazon S3 (Simple Storage Service): Provides centralized storage for raw sensor data, processed feature datasets, and model artifacts. S3's scalability ensures the system can handle growing data volumes as more equipment is monitored, while its versioning capabilities maintain data lineage and enable rollback when needed.

Amazon EC2 (Elastic Compute Cloud): Hosts the core processing infrastructure including data preprocessing scripts, machine learning model training and inference, and the Flask-based dashboard web application. EC2's flexibility allows for computational scaling as data volumes and model complexity increase over time.

AWS IAM (Identity and Access Management): Manages secure access to S3 buckets and EC2 instances, ensuring that only authorized components can read raw data, write processed results, or access model predictions. This security layer is critical for industrial applications handling sensitive operational data.

4.2.3 Automation and Scheduling

CRON Job Automation: Scheduled CRON jobs on EC2 automatically trigger data preprocessing and model inference at regular intervals (e.g., daily). These jobs read raw sensor data from S3, apply feature engineering transformations, generate health rating predictions, and store results back to S3. This eliminates manual intervention and ensures consistent, timely updates.

Dashboard Refresh: The dashboard employs the APScheduler library to periodically fetch the latest processed data and model predictions from S3. This ensures users always see current equipment health status without manual refresh, with configurable update intervals (e.g., hourly) to balance data freshness with system load.

4.2.4 Pipeline Benefits

The automated AWS architecture delivers several key advantages:

- **Efficiency:** Eliminates manual data handling and reduces human error by automating the entire pipeline from data ingestion through prediction generation.
- **Scalability:** Leverages AWS infrastructure to easily handle additional equipment or increased data frequency by scaling compute resources and storage as needed.
- **Reliability:** Scheduled jobs ensure consistent, timely updates to health predictions, reducing the risk of missed maintenance windows due to data processing delays.
- **Accessibility:** Centralized data storage in S3 and a web-based dashboard make equipment insights available to all stakeholders across different locations and devices.
- **Integration:** The modular architecture facilitates easy addition of new analytics or machine learning models as requirements evolve, without disrupting existing operations.

4.2.5 Current Implementation

The current pipeline processes data from three equipment types (tube mills, belt conveyors, high-temperature fans) with 15 sensor monitoring points generating measurements every 5 seconds. Health ratings are synchronized with sensor data every 20 minutes, and the complete feature engineering and prediction cycle runs automatically on a daily schedule. The dashboard provides real-time access to current equipment status, historical trends, and model prediction confidence intervals.

5 Conclusions and Recommendations

5.1 Conclusions

Brilliant Automation's transition from opaque MATLAB heuristics to our interpretable, automated machine learning pipeline has delivered on the core objectives of transparency, accuracy, and operational efficiency. By synchronizing high-frequency sensor streams with human-annotated health ratings, enriching the data with spectral and time-domain features, and deploying a suite of tailored models, we have achieved reliable predictions across 12–15 health metrics. The resulting dashboard and AWS-hosted pipeline now provide maintenance teams with real-time insights and explainable decision rules, improving trust and reducing manual processing by over 90%.

5.2 Limitations

Despite these advances, several challenges remain:

- **Predictability Gaps:** Metrics with inherently low signal correlation (e.g., `rubbing_condition_rating`) still yield modest R^2 scores, indicating the need for additional sensors or advanced anomaly detection.
- **Lack of diversity in target features:** Most ratings cluster in the high range (healthy operation), limiting variance and making it difficult for models to learn nuanced failure patterns.
- **Data Coverage:** With only seven days of initial data, some models exhibit sensitivity to seasonal or operational shifts not captured in the training period.
- **Retraining Overhead:** Current model retraining is scheduled manually; adapting to continuous inflows of new data requires further automation.
- **Complexity vs. Interpretability:** While tree ensembles deliver strong performance, their logic can be harder to convey than linear or rule-based models in certain operational contexts.Recommendations

5.3 Recommendations

- **Enhanced Monitoring & Alerts:** Integrate AWS CloudWatch alarms and event-driven Lambda functions to automate pipeline health checks, trigger real-time notifications for failures or anomalies, and visualize system status.
- **Serverless Processing:** Migrate data processing tasks to AWS Lambda and Kinesis for scalable, event-driven workflows that reduce operational costs and latency while simplifying infrastructure management.
- **CI/CD Pipeline:** Establish automated testing and deployment for preprocessing, feature engineering, and model training code using AWS CodePipeline or GitHub Actions to ensure consistency, reproducibility, and rapid iteration.
- **Extended Data Collection:** Expand the dataset by aggregating additional historical sensor readings and health ratings over longer operational periods and across similar equipment. More diverse data will improve model robustness, reduce overfitting, and capture rare failure modes.
- **Model Robustness Improvements:** Focus on advanced model techniques, such as unsupervised anomaly detection (e.g., Isolation Forest, Autoencoders) and periodic hyperparameter re-optimization, to improve prediction accuracy for low-variance ratings and adapt to evolving equipment behaviors.

By following these recommendations, Brilliant Automation can further solidify its leadership in transparent, data-driven equipment health monitoring and scale the solution across diverse industrial environments.

6 Acknowledgments

We thank Brilliant Automation Technology for their collaboration, and the maintenance staff for their domain insights. Our work reflects a joint effort across disciplines.

Team Contributions:

- Samuel Adetsi: Data analysis and preprocessing
- Mu Ha: Model development and evaluation
- Cheng Zhang: Pipeline engineering and infrastructure
- Michael Hewlett: Dashboard and interface design