Anomaly Detection for Predictive Maintenance

Jason Sonith, Steve Nguyen
University of South Alabama

Why This Matters

- Unplanned machine failures costs hundreds of thousands per hour
- Small companies and plants can't afford unexpected downtime
- But they also lack the resources needed for modern ML solutions:
  - ML specialists
  - GPU Servers
  - Labeled failure data

So the problem becomes:

"How do you bring predictive maintenance to teams who can't afford it?"

What’s Wrong With Current Solutions?

Deep learning models:

- Require GPU’s
- Hard to deploy
- Need large labeled datasets

Threshold alarms:

- Too many false alerts
- Don’t adapt to changes

Vendor Solutions:

- ❏ Expensive
- ❏ Cloud-locked
- ❏ Security concerns
- ❏ Exclusivity

Our Solution

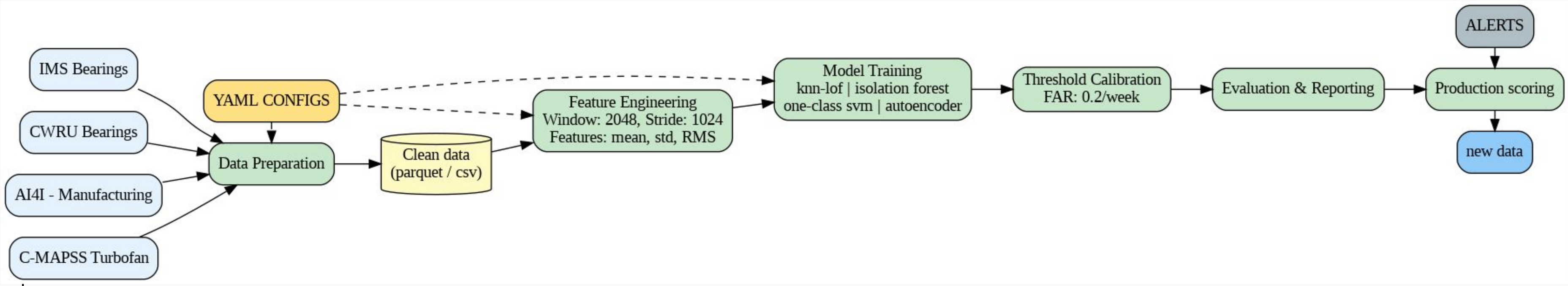
Our approach:

A lightweight, CPU-only, YAML-driven anomaly detection pipeline.

This pipeline:

- ❏ Runs on regular laptops
- ❏ No machine learning expertise required
- ❏ Uses simple YAML configs
- ❏ Calibrates false alarms per week
- ❏ 4 anomaly detection models
- ❏ SHAP explainability
- ❏ Works across 4 industrial datasets

Pipeline Architecture



Pipeline Stages:

- Data Prep
- Feature Engineering
- Model Training
- Threshold Calibration
- Evaluation & SHAP
- Production Scoring

How it works

Define in YAML	All settings in one file	Run Pipeline

```yaml
dataset_name: ims
paths:
  raw_input_dir: data/raw/ims/

window:
  size: 2048        # Window length
  stride: 1024      # 50% overlap

split:
  train_ratio: 0.60 # First 60% = healthy
  test_ratio: 0.30  # Last 30% = degraded

computed_features:
  - rms             # Root Mean Square
  - peak_to_peak    # Amplitude range
  - kurtosis        # Tail heaviness
```

```
# Data preparation
python scripts/prep_data.py --config configs/ims.yaml

# Feature engineering
python scripts/make_features.py --config configs/ims.yaml

# Train model
python scripts/train.py --config configs/models/isolation_forest.yaml

# Calibrate threshold
python scripts/threshold.py --target_far 0.2/week
```

```
Loading config from configs/ims.yaml
Dataset: ims
Loading 2156 files...
Loaded 44154880 rows
Columns with NaN values before dropping:
Dropped missing values: 44154880 rows remaining
Saving to data/clean/ims/ims_clean.parquet (Parquet)...
Saved 44154880 rows to parquet
Saving to data/clean/ims/ims_clean.csv (CSV)...
Saved 44154880 rows to csv
```

The Datasets

Datasets Used

IMS Bearings — real vibration until failure
CWRU Bearings — seeded bearing faults
AI4I — synthetic manufacturing cycles
NASA C-MAPSS — turbofan engine degradation

These Datasets Covers

Vibration, tabular, and multivariate time-series data.

Modeling Approach

4 Anomaly Detection Models

Isolation Forest

Fast & Robust

One-Class SVM

Normal boundary

Local Outlier Factor

Density-Based

Autoencoder

Reconstruction-based

Why SHAP is Important to Our Model

What is SHAP?

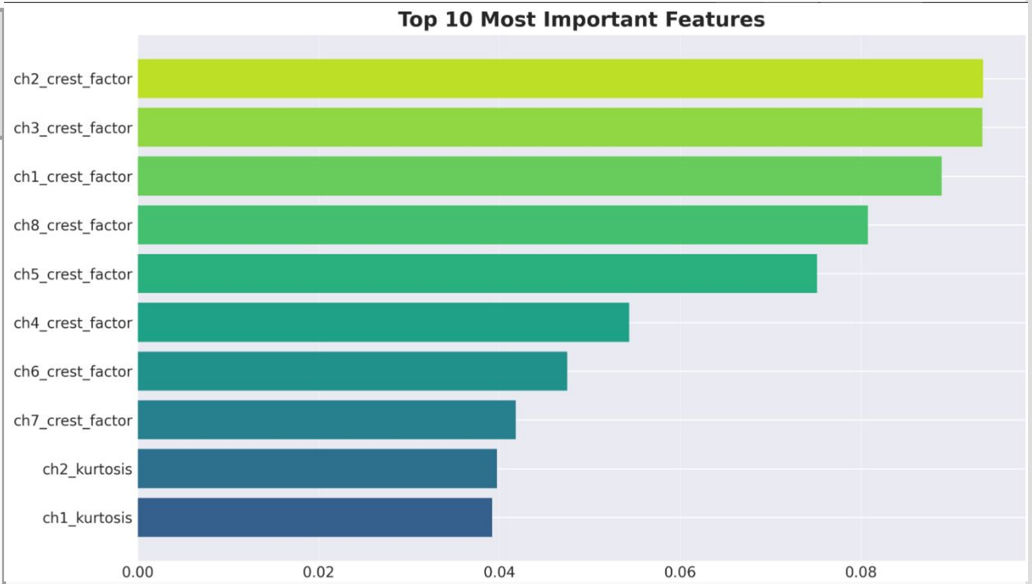
SHAP is an interpretability framework based on Shapley values that quantifies how each feature contributes to a model's prediction, indicating exactly which inputs increased or decreased the output.

- Explains feature influence: Shows the contribution of each input feature to the prediction.
- Improves model trust: Ensures the model bases decisions on meaningful physical signals rather than noise.
- Validates correctness: Confirms the model is learning the right patterns.
- Supports real-world deployment: Helps detect data leakage, overfitting, or unexpected model behavior before real-world use.
- Communicates results clearly: SHAP visualizations make model decisions easy to explain to non-technical audiences

Influence of SHAP on our scores

ML models are black boxes, Engineers don’t trust mysterious alerts

Alerts without SHAP	**Alert with SHAP**
❑ Bearing 3 failed: ❑ <u>Why</u> did it fail? ❑ <u>Where</u> did it fail? ❑ Do we <u>trust</u> the alert?	❑ Bearing 3 failed: ❑ <u>Why</u>: Channel 2 kurtosis spiked ❑ <u>Where</u>: Sensor on channel 2 ❑ <u>Trust</u>: We can verify the logic



Threshold Calibration

THRESHOLD CALIBRATION RESULTS

Model/Dataset	Target FAR	Estimated FAR	Threshold
IMS IForest	1.0/wk	0.989/wk	0.4851
IMS AutoEncoder	0.2/wk	0.200/wk	0.0137
IMS kNN-LOF	0.2/wk	0.200/wk	1.7821
IMS OC-SVM	2.0/wk	2.000/wk	-0.3182
AI4I IForest	0.2/wk	0.202/wk	0.4863
CWRU IForest	0.2/wk	0.212/wk	0.4795
FD001 IForest	0.2/wk	0.289/wk	0.4912
FD002 IForest	0.2/wk	0.216/wk	0.5025
FD003 IForest	0.2/wk	0.217/wk	0.4933
FD004 IForest	0.2/wk	0.221/wk	0.5016

Goal

Turn continuous scores into reliable alerts:

FAR - False Alarm Rate

(alerts per week)

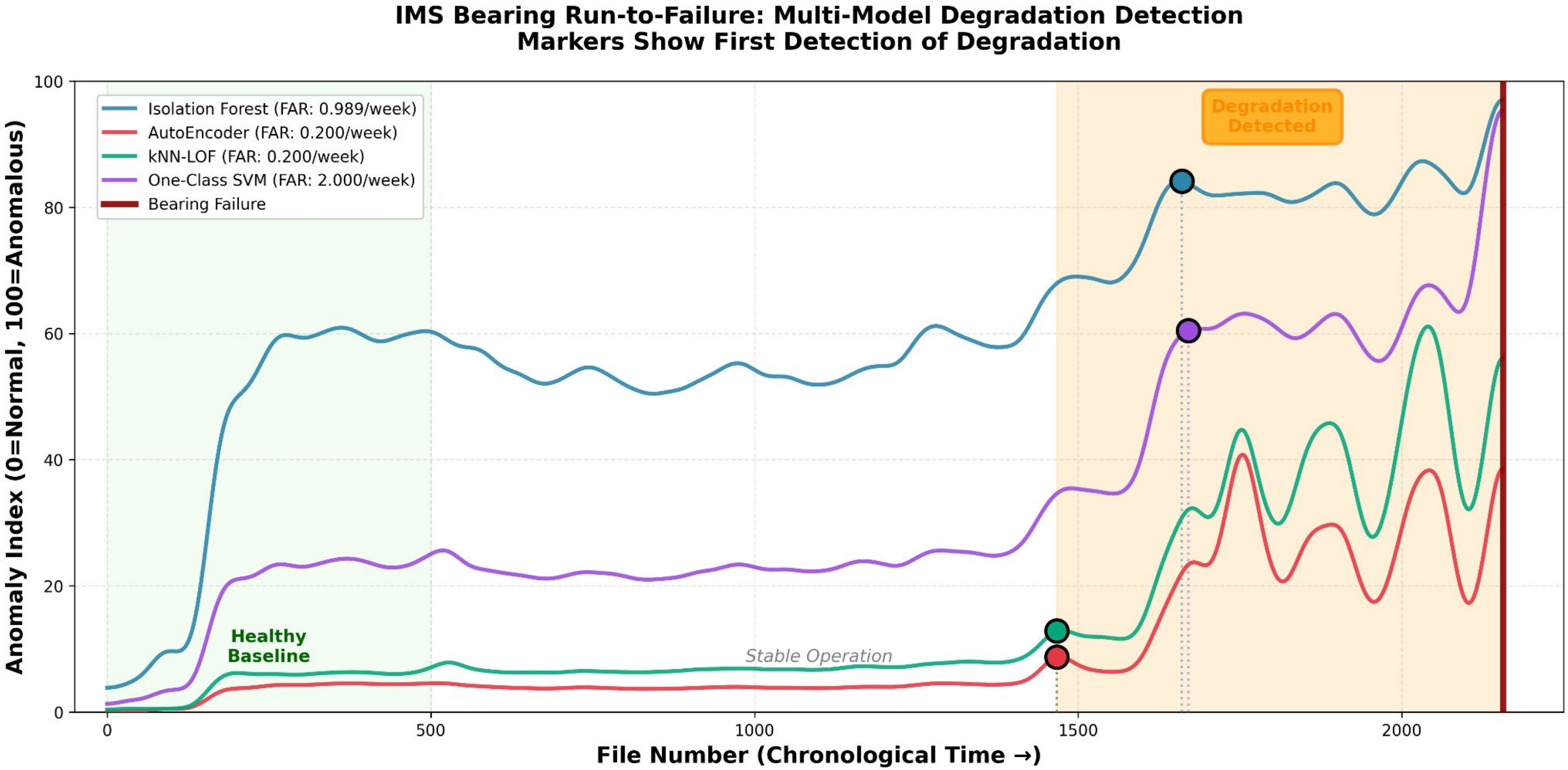
Calibrate

Compute threshold based on validation percentiles.

We aligned with ISA-18.2 alarm standards.

Results

Calibration Accuracy: 93-100%


IMS Bearing Run-to-Failure: Multi-Model Degradation Detection
Markers Show First Detection of Degradation
Isolation Forest (FAR: 0.989/week)
AutoEncoder (FAR: 0.200/week)
kNN-LOF (FAR: 0.200/week)
One-Class SVM (FAR: 2.000/week)
Bearing Failure
Degradation Detected
Healthy Baseline
Stable Operation
Anomaly Index (0=Normal, 100=Anomalous)
File Number (Chronological Time →)
0
20
40
60
80
100
0
500
1000
1500
2000

Highlights

Dataset	Results	Challenges
IMS	❑ **All models separated normal vs degraded states** ❑ **Isolation Forest had the best overall speed + stability**	❑ No labels ❑ Non-stationary vibration data
CWRU	❑ **ROC-AUC: 0.942** ❑ **PR-AUC: 0.964** ❑ **Precision: 1.000**	❑ Low recall due to strict FAR ❑ Seeded faults ≠ gradual failures
Overall-Cross	❑ **Thresholds consistent (~0.48 - 0.50)** ❑ **FAR stayed ~0.202 - 0.289/week** ❑ **Isolation Forest generalizes best**	❑ Windowing/Scaling differed per dataset ❑ Calibration depends on validation size

Insights and Limitations

Aspect	Explanation	Insight ✓	Limitation ✗
Config-Driven Design	YAML control made the pipeline reusable and easy to adapt without code changes.	✓	
Model-Agnostic Framework	The pipeline worked consistently across models because engineering mattered more than algorithm choice.	✓	
False-Alarm Control is Crucial	Industrial systems care more about predictable alert behavior than maximizing recall	✓	
Real-World Drift Not Modeled	Datasets don't capture long-term sensor movement or changing machine behavior.		✗
Limited Validation Sets	Small validation splits make threshold calibration less stable in some scenarios.		✗
Real-Time Monitoring	Pipeline only handles batch scoring and not real-time conditions.		✗
Autoencoder Complexity	Neural model required more compute + tuning compared to classical methods.		✗

Conclusion / Q&A

- ❑ We developed a lightweight, CPU-only anomaly detection pipeline for predictive maintenance on industrial equipment..
- ❑ A config-driven design allows easy reuse across machines, datasets, and sensors with minimal changes.
- ❑ False-alarm calibration enables predictable alert behavior aligned with industrial standards.
- ❑ The pipeline generalizes across four diverse datasets, demonstrating broad applicability.
- ❑ Classical models + strong engineering proved sufficient for reliable performance.
- ❑ Provides a practical foundation for deployable, real-world maintenance monitoring

Questions?