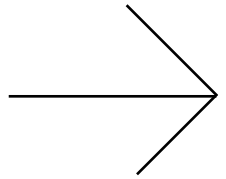
Rotating Machinery Health Monitoring System –
A Case study for ML based AI implementation

https://www.seekerssignpost.com/



Rev1.0 June 2026

Case Study :
– ML based Rotating Machinery Health Monitoring

Rotating machinery – motors, pumps, compressors, fans – are the workhorses of industrial plants, manufacturing facilities, and power generation systems. Unplanned failure of these machines causes production loss, safety risks, and expensive emergency repairs.

The engineering goal of the health monitoring system is to monitor the health of the machine continuously and detect problems early – before failure occurs.

The Machine:
A three-phase induction motor driving an industrial pump – representing one of the most common rotating machinery combinations in any industrial facility.

Failure modes we need to detect:

Failure Mode	How it develops	Consequence if missed
Bearing wear — inner race	Gradual surface degradation under load	Bearing seizure — machine stops
Bearing wear — outer race	Gradual surface degradation under load	Bearing seizure — machine stops
Shaft misalignment	Installation error or thermal expansion	Accelerated bearing and seal wear
Rotor imbalance	Mass distribution asymmetry — blade loss, deposit buildup	Excessive vibration — structural damage
Looseness	Mechanical looseness in mounting or coupling	Progressive structural damage
Lubrication failure	Oil film breakdown — starvation or contamination	Rapid bearing destruction

Example Sensor Suite: – For Rotating Machinery Health Monitoring

Five sensor channels. Six failure modes. Five decisions. This is the problem we will now solve — first the traditional way, then with ML.

Sensor	Qty	Placement	What it measures
RTD temperature sensor	3	Front bearing, rear bearing, machine body	Bearing and winding thermal condition
Analog accelerometer	1	Bearing housing – radial direction	Mechanical vibration – amplitude and frequency
Acoustic emission sensor	1	Bearing housing	High frequency stress waves from surface defects

What intelligent decisions are needed:

- Is the machine operating normally right now?
- Is there an early sign of a developing fault?
- Which specific failure mode is developing?
- How much time remains before maintenance is required?
- Is immediate shutdown necessary?

Traditional Approach – FFT Based Decision Making

Traditional Approach – Solving It With Signal Processing and Domain Knowledge

Before ML existed as a practical tool – and still today in many industrial installations – rotating machinery health monitoring was solved using **Fast Fourier Transform (FFT) analysis** combined with deep domain knowledge of rotating machinery physics. This is a mature, well understood, and effective engineering approach.

The Core Principle:
Every mechanical fault in a rotating machine produces a characteristic vibration signature at a predictable frequency. These frequencies are mathematically derivable from the machine's physical parameters.

Fault Characteristic Frequencies – Derivable From Machine Parameters:

Fault	Characteristic Frequency	Derived From
Bearing inner race defect	BPFI = (N/2) × RPM/60 × (1 + Bd/Pd × cosα)	Number of balls, ball diameter, pitch diameter, contact angle
Bearing outer race defect	BPFO = (N/2) × RPM/60 × (1 − Bd/Pd × cosα)	Same bearing geometry parameters
Shaft misalignment	2 × RPM/60	Shaft running speed
Rotor imbalance	1 × RPM/60	Shaft running speed
Mechanical looseness	0.5 × RPM/60 and harmonics	Sub-harmonic of running speed
Gear mesh (if present)	Number of teeth × RPM/60	Gear parameters

The Traditional Workflow:

- Collect accelerometer signal continuously — time domain waveform
- Apply FFT — convert time domain signal to frequency domain spectrum
- Calculate expected fault frequencies from known machine parameters
- Compare FFT spectrum peaks against expected fault frequencies
- Apply amplitude thresholds — if peak at bearing frequency exceeds threshold — fault detected
- Correlate with RTD temperature data — rising bearing temperature confirms developing fault
- Correlate with acoustic emission — high frequency bursts confirm surface defect activity

What the Engineer Sees:
What temperature data adds:

Observation in FFT Spectrum	Diagnosis
Peak at 1× running speed — elevated	Rotor imbalance
Peak at 2× running speed — elevated	Shaft misalignment
Peak at BPFO frequency — elevated	Outer race bearing defect
Peak at BPFI frequency — elevated	Inner race bearing defect
Sub-harmonics at 0.5× — present	Mechanical looseness
Broadband noise floor rising	Lubrication degradation

RTD Reading	Interpretation
Front bearing temperature rising	Front bearing developing fault
Rear bearing temperature rising	Rear bearing developing fault
Both bearings rising together	Lubrication failure or overload
Body temperature rising alone	Winding issue — electrical fault
Temperature differential increasing	Misalignment causing uneven load

Strengths of This Approach:

- Physically grounded — every decision traceable to a known equation
- No data collection or training required — works from day one
- Explainable — the engineer can point to the exact FFT peak causing the alarm
- Mature — decades of industrial validation and standards exist
- Deterministic — same input always produces same output
- Certifiable — well understood failure mode analysis path

This approach works excellently when faults are developed enough to produce a clear frequency signature. The question is — what does it miss, and when does that matter? That is the subject of the next slide.

Is the Traditional Approach Enough?

Is the FFT Based Approach Enough? – An Honest Engineering Assessment

The FFT based approach is not wrong – it is proven, mature, and effective. The question is not whether to replace it – the question is where it has limitations and whether those limitations matter for your specific application.

Where FFT Based Monitoring Works Excellently:

Scenario	**Why FFT works well**
Developed bearing faults	Clear frequency peak visible in spectrum – easy to detect
Rotor imbalance	Strong 1× peak – unmistakable signature
Shaft misalignment	Strong 2× peak – well established diagnosis
Steady state operation	Machine runs at constant speed – FFT frequencies are stable and predictable
Known machine parameters	Bearing geometry known – fault frequencies precisely calculable
Experienced analyst available	Domain expert interprets spectrum – high accuracy diagnosis

FFT Based Monitoring - Limitations:

Limitation	Why it matters
Early stage fault detection	A developing bearing defect produces a very weak frequency signature – buried in noise – FFT misses it until fault is already significant
Variable speed machines	Fault frequencies shift with RPM – FFT spectrum smears – order tracking needed – adds complexity
Compound faults	Two or more faults developing simultaneously – spectra overlap – diagnosis becomes ambiguous
Intermittent faults	Fault appears and disappears – FFT on a fixed window misses transient events
Novel fault signatures	A fault not in the engineer's knowledge base – no known characteristic frequency – missed entirely
Subtle degradation trends	Gradual change in noise floor or spectral shape – human analyst may not notice until significant
Continuous 24/7 monitoring without expert	Automated threshold alarms generate false positives – nuisance alarms – operators start ignoring them

The Critical Gap — Early Detection:

The most important limitation is early fault detection. Consider bearing degradation as it develops:

Stage	**What is happening physically**	**FFT visibility**	**Consequence**
Stage 1 — Incipient	Microscopic surface fatigue beginning	Not visible in FFT	No alarm — fault developing undetected
Stage 2 — Early	Small surface defects forming	Weak signal — buried in noise	Marginal — may miss
Stage 3 — Moderate	Clear defect — impacts generating	Clear FFT peak	FFT detects reliably
Stage 4 — Advanced	Significant material loss	Strong FFT peak with harmonics	FFT detects easily
Stage 5 — Failure imminent	Bearing structurally compromised	Very strong signal	Too late for planned maintenance

FFT reliably catches Stage 3 and beyond. Stage 1 and 2 — where planned maintenance intervention has the most value — are where FFT struggles.

So – Do We Need ML?

The honest engineering answer is — it depends:

Situation	Recommendation
Non-critical machine – failure acceptable – reactive maintenance strategy	FFT approach is sufficient – proven and simple
Important machine – planned maintenance valuable – moderate consequence of failure	FFT as primary – ML anomaly detection as early warning layer
Critical machine – failure catastrophic – maximum advance warning needed	ML based approach essential – FFT as validation layer
Variable speed machine – complex fault patterns – multiple simultaneous faults possible	ML justified – FFT alone insufficient
Large fleet of identical machines – data abundant – ROI on ML development justified	ML strongly justified – fleet data accelerates training

The Engineering Conclusion: *FFT based monitoring and ML based monitoring are not competitors — they are complementary layers. FFT provides physically grounded, explainable, certifiable diagnosis of developed faults. ML provides early warning of developing anomalies before they become visible in the frequency spectrum. The right question is not FFT or ML — it is at what stage of fault development do you need to intervene, and what is the consequence of missing an early warning?*

ML based AI Implementation for Rotating Machinery Health Monitoring System

End-to-End Implementation Flow (Using STM32 MCUs & Ecosystem – Without NanoEdge AI Studio)

1 Define Problem & System

Goal: Predict machine condition in real time

Outputs (example):
- Normal
- Misalignment
- Bearing Fault
- Imbalance
- Lubrication Issue
- Degradation
- Critical Fault

2 Instrument the System

3x RTD Temperature Channels (Front Bearing, Back Bearing, Machine Body)

1x Analog Accelerometer (Vibration)

1x Acoustic Sensor (Structure-borne Sound)

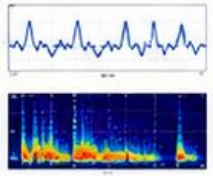
Signal Conditioning & Anti-alias Filtering

3 Data Acquisition & Logging

- STM32 ADC + DMA
- High speed sampling for vibration & acoustic
- Timestamp, RPM/Load
- Store raw data (binary/CSV)

Build a diverse dataset across operating conditions & faults

4 Data Preprocessing & Feature Engineering



- Filter, Resample
- Windowing (e.g., 1s with 50% overlap)
- Extract Features
  - Temperature: mean, std, gradients, ΔT
  - Vibration: RMS, Peak, Kurtosis, Band Energy, Envelope, etc.
  - Acoustic: FFT bins, Spectral features, MFCCs, etc.

Create feature vector for each window

5 Labeling (Ground Truth)

- Normal
- Misalignment
- Bearing Fault
- Imbalance
- Lubrication Issue
- Degradation
- Critical Fault

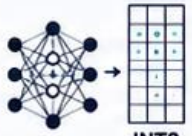
Label each window with machine condition

6 Model Training (Offline)

- Use Python (scikit-learn / TF)
- Algorithms:
  - Random Forest
  - SVM
  - Shallow Neural Network (MLP)
- Evaluate with accuracy, precision, recall, F1-score, confusion matrix

Select best performing model

7 Model Optimization & Quantization



- Convert model to TensorFlow Lite (.tflite)
- Quantize to INT8 for embedded deployment
- Target small model size & low memory footprint

Ensure accuracy retention after quantization

8 Model Conversion with STM32Cube.AI

- Import .tflite model
- Optimize for target STM32 MCU
- Generate C inference code & header files
- Get memory & performance report
- Validate model output

Ready-to-use embedded C code for inference

9 Application Development on STM32

Using STM32CubeMX & STM32CubeIDE

- **Configure peripherals** (ADC, DMA, I2C/SPI, Timers, UART, etc.)
- **Integrate X-CUBE-AI middleware**
- **Integrate CMSIS-DSP** for signal processing
- **Build application firmware**

Compile & Flash to STM32 MCU

10 Real-time Inference & Decision Logic

On STM32 MCU (in production)

- Acquire sensor data
- Preprocess & extract features
- Normalize features
- Run ML inference
- Apply decision logic with thresholds & persistence
- Generate health status & alarms

Decision in microseconds on every window

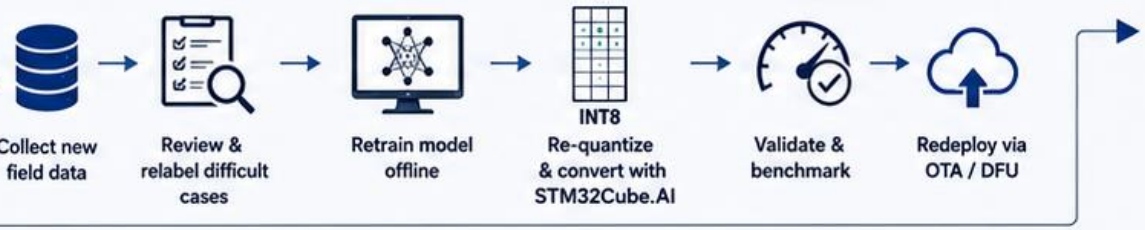
11 Deployment, Monitoring & Field Operation

UART / CAN / RS485 / Ethernet (MQTT)

- Send health status & alarms
- Visualize on dashboard (STM32CubeMonitor or custom HMI)
- Log events
- OTA / DFU updates for firmware & model

Continuous monitoring in the field

12 Continuous Improvement Loop



Benefits

- ✓ Early fault detection & reduced downtime
- ✓ Lower maintenance cost
- ✓ Improved safety & reliability
- ✓ Runs fully on STM32 MCU (edge intelligence)
- ✓ Scalable & continuously improving

Key ST Tools Used

- STM32CubeMX – Configuration & Code Generation
- STM32CubeIDE – Development Environment
- STM32Cube.AI – Model Conversion & Optimization
- STM32CubeMonitor – Monitoring & Visualization
- CMSIS-DSP – Signal Processing Library

ⓘ This flow enables an end-to-end ML based AI solution for rotating machinery health monitoring using STM32 MCUs and the ST ecosystem.

ML based AI Implementation for Rotating Machinery Health Monitoring System

End-to-End Implementation Flow (Using NanoEdge AI Studio)

- **Faster Development** with no/low-code ML
- **Optimized for STM32** from edge to cloud
- **Small Footprint** & Low Power
- **Built-in Anomaly Detection** & Classification
- **Easy Update** & Redeploy

Use Case Sensors

- 3x RTD Temperature (Front, Back Bearing, Body)
- 1x Analog Accelerometer (Vibration)
- 1x Acoustic Sensor (Structure-borne Sound)

1 Define Problem & System

Goal: Predict machine condition in real time

Outputs (example):

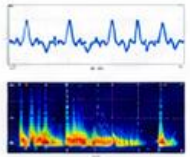
- Normal
- Misalignment
- Bearing Fault
- Imbalance
- Lubrication Issue
- Degradation
- Critical Fault

2 Instrument the System & Acquire Data

- 3x RTD Temperature Channels
- 1x Analog Accelerometer (Vibration)
- 1x Acoustic Sensor (Structure-borne Sound)

- STM32 ADC + DMA
- High speed sampling
- Timestamp, RPM/Load
- Store raw data (binary/CSV)

3 Prepare Data for Training



- Filter, Resample, Denoise
- Windowing (e.g., 1s with 50% overlap)
- Extract Features
  - Time, Frequency, Time-Frequency
  - Temperature: mean, std, gradients, ΔT
  - Vibration: RMS, Peak, Kurtosis, Band Energy, etc.
  - Acoustic: FFT bins, Spectral features, MFCCs, etc.

4 Label Data (Ground Truth)

- Normal
- Misalignment
- Bearing Fault
- Imbalance
- Lubrication Issue
- Degradation
- Critical Fault

Label each window with machine condition

5 Train with NanoEdge AI Studio

NanoEdge AI Studio

- Import dataset (features or signals)
- Select task:
  - Classification or Anomaly Detection
- Configure training
- Train & validate
- Auto model search (optional)

High accuracy with minimal effort

6 Model Evaluation & Selection

- Evaluate with built-in metrics
- Confusion matrix, F1-score, recall, precision
- Select best model based on accuracy & footprint

Choose model that meets performance target

7 Optimize for STM32

- Automatic optimization for target MCU
- Quantization (INT8)
- Pruning
- Memory & latency estimation

Small, fast and power-efficient model

8 Generate Deployment Package

- Generate optimized C code library
- Includes model weights, inference engine & APIs
- Model + metadata + performance report

Ready-to-use embedded package

9 Application Development on STM32

STM32 MCU

- Use STM32CubeMX & STM32CubeIDE
- Configure peripherals (ADC, DMA, I2C/SPI, Timers, UART, etc.)
- Integrate NanoEdge AI library
- Implement data pipeline & inference

Build firmware with NanoEdge AI library

10 Real-time Inference & Decision Logic

- Run inference on STM32 MCU
- Decision logic with thresholds & persistence
- Generate health status & alarms

Decision in microseconds on every window

11 Deployment, Monitoring & Field Operation

- Send health status & alarms
- Visualize on dashboard (STM32CubeMonitor or custom HMI)
- Log events
- OTA / DFU updates for firmware & model

Continuous monitoring in the field

12 Continuous Improvement Loop

Collect new field data → Review & relabel difficult cases → Retrain & improve model in NanoEdge AI Studio → Re-optimize for target STM32 → Redeploy via OTA / DFU

- ✓ Continually adapts to new operating conditions
- ✓ Improves accuracy over time
- ✓ Lower downtime & maintenance cost
- ✓ Higher reliability & safety

Key ST Ecosystem Tools

- STM32CubeMX
- STM32CubeIDE
- STM32CubeMonitor
- Monitoring & Visualization
- X-CUBE-AI Deployment Middleware
- CMSIS-DSP Signal Processing Library
- STM32 MCUs Optimized for Edge AI

ⓘ This flow enables an end-to-end ML based AI solution for rotating machinery health monitoring using STM32 MCUs and NanoEdge AI Studio.

The ML Approach — What Data Do We Need?

Before selecting an algorithm, before thinking about training, before considering deployment — the first and most important question in any ML project is:

What data does this system generate — and what do we need to label?
What Our Five Sensor Channels Produce:

Sensor Channel	Data Type	Sampling Rate	What it captures
RTD — Front bearing	Temperature — scalar value	1 sample per second	Thermal condition of front bearing
RTD — Rear bearing	Temperature — scalar value	1 sample per second	Thermal condition of rear bearing
RTD — Machine body	Temperature — scalar value	1 sample per second	Overall thermal condition
Accelerometer	Vibration — time domain waveform	10,000 to 50,000 samples per second	Mechanical vibration — all frequency content
Acoustic emission	Stress wave — time domain waveform	100,000 to 1,000,000 samples per second	High frequency surface defect activity

Raw Data is Not What ML Consumes Directly:

- The accelerometer and acoustic sensor produce high speed waveforms. Feeding raw waveforms directly to a simple ML algorithm is inefficient and demands far more data and compute than necessary.

- The engineering approach is to extract **features** — meaningful numerical summaries computed from the raw waveform — and feed those features to the ML algorithm.

Feature Extraction – Converting Raw Signals to Structured ML Input:

Feature	Computed From	What it represents
RMS amplitude	Accelerometer waveform	Overall vibration energy level
Peak amplitude	Accelerometer waveform	Maximum vibration excursion
Crest factor	Peak / RMS	Impulsiveness – early indicator of bearing impacts
Kurtosis	Accelerometer waveform	Statistical impulsiveness – sensitive to early bearing defects
Skewness	Accelerometer waveform	Signal asymmetry – looseness indicator
FFT amplitude at 1× RPM	FFT of accelerometer	Imbalance energy
FFT amplitude at 2× RPM	FFT of accelerometer	Misalignment energy
FFT amplitude at BPFO	FFT of accelerometer	Outer race bearing defect energy
FFT amplitude at BPFI	FFT of accelerometer	Inner race bearing defect energy
Acoustic RMS	Acoustic emission waveform	Overall surface defect activity
Acoustic peak	Acoustic emission waveform	Impulsive surface defect events

The Complete Feature Vector Fed to ML:

Every measurement cycle produces one feature vector – a row of numbers representing the machine state at that moment:

[Front_Temp, Rear_Temp, Body_Temp, Vib_RMS, Vib_Peak,
Crest_Factor, Kurtosis, Skewness, FFT_1x, FFT_2x,
FFT_BPFO, FFT_BPFI, AE_RMS, AE_Peak]

= 14 numbers representing machine health at one point in time

This feature vector is structured, compact, and physically meaningful. The ML algorithm learns the boundaries between healthy and unhealthy regions in this 14-dimensional space.

What Does Labeling Mean Here?

A label is the answer to the question — what was the machine's true condition when this feature vector was recorded?

Label	Meaning	How it is assigned
Normal	Machine is healthy – all systems operating correctly	Engineer confirms during routine inspection
Anomaly	Something has changed – not yet diagnosed	ML flags deviation – engineer investigates
Imbalance	Rotor imbalance confirmed	Maintenance engineer confirms after inspection
Misalignment	Shaft misalignment confirmed	Maintenance engineer confirms after correction
Bearing outer race	Outer race defect confirmed	Confirmed at bearing replacement
Bearing inner race	Inner race defect confirmed	Confirmed at bearing replacement
Lubrication fault	Lubrication failure confirmed	Confirmed by oil analysis or inspection

The Labeling Process in Practice:

- Normal data — collected continuously during confirmed healthy operation — abundant
- Anomaly labels — assigned when maintenance team investigates an ML flag and finds something
- Specific fault labels — assigned at the moment of maintenance intervention — bearing replaced, misalignment corrected, confirmed and recorded
- Every labeled event becomes a permanently valuable training example — stored with its full feature vector

The data collection process sounds straightforward. The challenge reveals itself when you ask — how many labeled fault examples can we realistically collect? That is the subject of the next slide.

The Labeled Data Problem — The Hardest Part of Industrial ML

- The algorithm is not the hard part. Quantization is not the hard part. Deployment is not the hard part.
- **Getting sufficient labeled fault data — that is the hardest part of industrial ML.**
- And it is a problem that no algorithm, no framework, and no amount of compute can solve alone.

Why Normal Data is Easy to get – Why Fault Data is Not:

Data Type	Availability	Why
Normal operation data	Abundant – thousands of hours	Machine runs normally most of its life
Imbalance data	Limited – controlled lab experiment possible	Can be induced artificially – but not always representative
Misalignment data	Limited – controlled experiment possible	Can be induced carefully in controlled conditions
Bearing wear – early stage	Very rare	Must wait for real bearing to develop real defect
Bearing wear – advanced stage	Rare – and dangerous to collect	Running a machine to bearing failure risks collateral damage
Lubrication failure data	Very rare – and dangerous	Oil starvation can destroy machine rapidly
Catastrophic fault data	Almost never available	Nobody deliberately destroys production equipment

The Class Imbalance Reality:

In a typical industrial deployment after one year of operation — a realistic labeled dataset might look like this:

Label	Labeled examples collected	Percentage of dataset
Normal	50,000 feature vectors	97.5%
Imbalance	800 feature vectors	1.5%
Misalignment	400 feature vectors	0.8%
Bearing outer race	100 feature vectors	0.2%
Bearing inner race	50 feature vectors	0.1%
Lubrication fault	20 feature vectors	0.04%
Total	**51,370**	**100%**

An ML algorithm trained on this dataset will learn to say "Normal" almost every time — because that is what 97.5% of its training experience told it. It will miss the faults that matter most.

Why Each Fault Type is Difficult to Label:

Bearing wear — the most common and most critical fault:

- A bearing does not fail suddenly — it degrades over weeks or months
- Early-stage defects produce no visible damage at inspection
- The exact moment a bearing transitions from healthy to unhealthy is not observable
- When the bearing is finally replaced — only the end state is confirmed — not the progression
- Early stage labeled examples are therefore almost impossible to obtain from real field events

Misalignment and imbalance:

- These can be induced in a controlled laboratory setting
- But laboratory induced faults on a test rig may not represent the exact signature of the same fault on your specific machine in your specific installation
- Transfer from lab data to field machine is imperfect

Compound faults:

- Real machines often develop multiple faults simultaneously
- A worn bearing on a misaligned shaft produces a combined signature
- Labeling compound faults requires expert analysis — not just maintenance confirmation

The Labeling Quality Problem:

Even when fault events occur — labeling quality is a challenge:

Labeling challenge	Why it happens	Consequence
Delayed confirmation	Fault confirmed weeks after it developed	Feature vectors from fault onset are labeled Normal — incorrect
Ambiguous diagnosis	Maintenance team disagrees on fault type	Incorrect or missing labels
Incomplete records	Maintenance event not recorded in system	Valuable labeled data lost permanently
Sensor data gap	Data logging failed during fault event	Event occurred but no feature vectors captured

The Honest Summary:

Challenge	Impact on ML project
Rare fault events	Insufficient labeled examples for reliable fault classification
Class imbalance	ML learns to predict Normal — misses critical faults
Dangerous fault induction	Cannot safely generate advanced fault data on production equipment
Labeling quality	Even available fault data may be incorrectly labeled
Early stage fault labeling	The most valuable data — early warning — is the hardest to label correctly

This is why the first deployment of ML on a rotating machinery system should never attempt full fault classification immediately. The data does not exist yet to support it. The engineering answer to this challenge is on the next slide.

Starting Smart – Learn Normal First – Train on Healthy Data Only

Given the labeled data problem – the practical and immediately deployable ML strategy for rotating machinery is:

Do not attempt to classify faults first. Learn what normal looks like – and detect anything that deviates from it.

This approach is called **Unsupervised Anomaly Detection** – and it requires only one label – Normal.

The Core Principle:

Instead of teaching ML the difference between fault types – teach ML the boundaries of healthy operation. Anything outside those boundaries is an anomaly – regardless of what type of fault it is.

TRAINING PHASE — Healthy machine only:

Feature vectors from healthy operation
→ ML learns the statistical signature of normal
→ ML builds a model of what normal looks like
→ No fault data needed — no fault labels needed

DEPLOYMENT PHASE — Live monitoring:

New feature vector arrives every measurement cycle
→ ML asks: does this look like normal?
→ Yes → No alarm
→ No → Anomaly flagged → Engineer investigates

What Normal Data Captures:

Normal operation is not a single fixed point — it is a region in the feature space that encompasses all valid healthy operating conditions:

Variation in normal operation	**What changes in feature vector**
Different load conditions	RMS amplitude varies with load
Ambient temperature changes	RTD baseline shifts seasonally
Speed variations within operating range	FFT frequency positions shift slightly
Startup and shutdown transients	Short duration elevated vibration – expected
Different operators – different loading patterns	Statistical distribution of features shifts

The ML model must learn all of these as normal — otherwise it generates nuisance alarms on every load change. This is why collecting normal data across all real operating conditions is critical before training.

Normal Data Collection Protocol:

Phase	Duration	What to capture	Purpose
Baseline collection	Minimum 4-6 weeks	All load conditions, all shifts, all ambient temperatures	Capture full envelope of healthy operation
Seasonal variation	Ideally 3 months+	Summer and winter operation if applicable	Capture thermal baseline variation
Operational events	Throughout	Startups, shutdowns, load changes	Teach model these are normal transients
Post maintenance	After any maintenance	Fresh bearing, corrected alignment	Establish new healthy baseline if machine changes

Algorithms for Unsupervised Anomaly Detection:

Algorithm	How it works	Best for	Runs on MCU?
Autoencoder Neural Network	Learns to reconstruct normal data — reconstruction error is anomaly score	Complex multi-sensor systems — captures non-linear relationships	Yes — quantized small autoencoder
Isolation Forest	Isolates anomalies by random partitioning — anomalies are isolated faster	Tabular feature vectors — fast training	Yes — lightweight
One-Class SVM	Builds a boundary around normal data in feature space	Small feature vectors — well understood mathematically	Yes — efficient inference
PCA based detection	Projects data to lower dimensions — reconstruction error flags anomalies	When features are correlated — dimensionality reduction useful	Yes — very lightweight
Statistical threshold on features	Mean and standard deviation of each feature — flag beyond N sigma	Simplest approach — individual feature monitoring	Yes — minimal compute

How Each Approach Behaves on Rotating Machinery:

Approach	Detects early bearing wear?	Detects misalignment?	Detects lubrication fault?	Runs on STM32 M4?
Statistical baseline	Partially – if RMS or kurtosis rises	Yes – if FFT 2x rises	Yes – if temperature rises	Yes – trivial
PCA detection	Better – captures correlated features	Yes	Yes	Yes – lightweight
Autoencoder	Best – captures non-linear relationships	Yes	Yes	Yes – after INT8 quantization
Isolation Forest	Good – robust to outliers	Yes	Yes	Yes – lightweight

Recommended Starting Point for Your System:

Given your 14-feature vector and STM32 class target hardware:

Decision	Recommendation	Reason
Algorithm	Autoencoder — small architecture	Captures relationships between all 14 features simultaneously
Architecture	Input 14 → Hidden 8 → Bottleneck 4 → Hidden 8 → Output 14	Small enough for MCU after quantization
Training	Workstation — PyTorch or TensorFlow	Standard training environment
Quantization	INT8 — STM32Cube.AI or CMSIS-NN	Fits in STM32 flash — runs efficiently
Anomaly threshold	Set at 99th percentile of training reconstruction error	Balances sensitivity against false alarm rate
Deployment	Via OTA, JTAG, DFU, or any supported update mechanism	Standard embedded deployment

What This System Gives You From Day One:

Every anomaly flag is a data collection opportunity:

- Engineer investigates the machine physically
- Inspection finding is recorded — what was found, what was done
- If a fault is confirmed — the feature vectors from that event are labeled with the specific fault type
- These labeled examples are stored — they will train the next model version
- If nothing found — nuisance alarm recorded — threshold or model adjusted

Every investigation — whether it finds a fault or not — makes the next model version smarter.

The system is now live, monitoring continuously, and collecting its own labeled fault data organically. The next slide shows how this evolves into full fault classification over time.

What Happens When an Anomaly is Flagged:

Every anomaly flag is a data collection opportunity:

- Engineer investigates the machine physically
- Inspection finding is recorded — what was found, what was done
- If a fault is confirmed — the feature vectors from that event are labeled with the specific fault type
- These labeled examples are stored — they will train the next model version
- If nothing found — nuisance alarm recorded — threshold or model adjusted

Every investigation — whether it finds a fault or not — makes the next model version smarter.

The system is now live, monitoring continuously, and collecting its own labeled fault data organically. The next slide shows how this evolves into full fault classification over time.

Growing Intelligence Over Time

Growing Intelligence Over Time — From Anomaly Detection to Fault Classification

- The unsupervised anomaly detection system deployed in Phase 1 is not the final destination — it is the starting point.
- As the system operates in the field, it organically accumulates the labeled fault data that was impossible to collect before deployment.
- Intelligence grows with operational time — and each redeployment makes the system more capable.

The Evolution — Three Phases:
Phase 1 — Anomaly Detection — Deploy Immediately

Aspect	Detail
What ML knows	What normal looks like — trained on healthy data only
What it can decide	Normal or Anomaly — binary output
Data needed to deploy	Normal operation data only — 4 to 6 weeks minimum
Value delivered	Early warning of any deviation — before fault is visible in FFT
Limitation	Cannot identify which fault is developing
Duration	Until sufficient labeled fault data accumulates — typically 6 to 18 months

How Labeled Fault Data Accumulates During Phase 1:

Every anomaly flag triggers an engineering response. That response generates labeled data:

Anomaly flagged by ML system

↓

Maintenance engineer investigates

↓

- Fault confirmed → Label assigned → Stored as training example
  Example: "Outer race bearing defect — confirmed at replacement"
- Early stage fault found → Monitored → Labeled as fault develops
  Example: "Bearing degradation Stage 2 — confirmed by acoustic emission"
- Nothing found → Nuisance alarm → Model threshold adjusted

Phase 2 — Fault Classification — When Data Justifies It

When sufficient labeled fault examples have accumulated — the model is retrained to classify specific fault types.

Aspect	Detail
What ML knows	Normal signature AND specific fault signatures
What it can decide	Normal / Imbalance / Misalignment / Bearing outer race / Bearing inner race / Lubrication fault
Data needed	Minimum 50 to 100 confirmed labeled examples per fault type
Value delivered	Specific fault identification — maintenance team knows exactly what to fix
Additional capability	Fault severity estimation — early / moderate / advanced
Duration	Until sufficient degradation trend data accumulates for RUL estimation

The Retraining and Redeployment Cycle:

Field deployment — Phase 1 anomaly detector running

↓

Labeled fault examples accumulate over time

↓

Sufficient data for new fault class → Retrain on workstation

↓

Validate new model — compare against previous version

↓

Quantize — INT8 — package into firmware

↓

Redeploy via OTA, JTAG, DFU, or supported update mechanism

↓

Phase 2 fault classifier running — continues accumulating data

↓

Repeat cycle — model improves with every redeployment

Phase 3 – Remaining Useful Life Estimation

The most valuable capability — and the hardest to achieve. Requires fault progression data — feature vectors labeled with fault severity at each stage of development.

Aspect	Detail
What ML knows	How faults progress from early stage to failure – the degradation trajectory
What it can decide	How much time remains before maintenance intervention is required
Data needed	Multiple complete fault progression sequences – from healthy through to confirmed failure
Value delivered	Planned maintenance at exactly the right time – maximum machine utilisation
Hardware requirement	Same MCU – model complexity similar to Phase 2
Timeline to achieve	Typically, 2 to 4 years of field operation on a fleet of machines

What Changes Between Phases — And What Does Not:

Aspect	Phase 1	Phase 2	Phase 3
ML approach	Unsupervised — anomaly detection	Supervised — fault classification	Supervised — regression for RUL
Labels needed	Normal only	Normal + specific fault types	Normal + fault severity progression
Output	Normal / Anomaly	Specific fault type	Remaining useful life — hours or days
Hardware on MCU	Same	Same	Same
Inference engine	Same	Same	Same
Model weights	Updated via redeployment	Updated via redeployment	Updated via redeployment
FFT still used?	Yes — as validation layer	Yes — as confirmation layer	Yes — as physical grounding

The Fleet Advantage:

The most valuable capability — and the hardest to achieve. Requires fault progression data — feature vectors labeled with fault severity at each stage of development.

Fleet size	Fault events per year	Labeled examples after 2 years
1 machine	2 to 4 events	4 to 8 labeled fault examples
10 machines	20 to 40 events	40 to 80 labeled fault examples
50 machines	100 to 200 events	200 to 400 labeled fault examples
100 machines	200 to 400 events	Sufficient for Phase 2 and Phase 3

This is why ML based machinery health monitoring delivers its greatest value when deployed across a fleet — not on a single machine. One model trained on fleet data benefits every machine in the fleet.

The Honest Timeline:

Milestone	Realistic timeline
Phase 1 deployed — anomaly detection live	Month 2 to 3 from project start
First labeled fault examples accumulating	Month 3 onwards
Sufficient data for Phase 2 — single machine	12 to 24 months
Sufficient data for Phase 2 — fleet of 50	3 to 6 months
Phase 3 RUL estimation — single machine	3 to 5 years
Phase 3 RUL estimation — fleet of 50	12 to 18 months

Patience and operational discipline — recording every maintenance event, every anomaly investigation, every fault confirmation — is what determines how quickly the system matures. The algorithm is the easy part. The data discipline is everything.

Standing on Others' Shoulders

Standing on Others' Shoulders – Public Datasets and Transfer Learning

The labeled data problem does not mean starting from zero. The research community and industry have made high quality rotating machinery fault datasets publicly available. These datasets contain what is hardest to collect in the field – controlled, well labeled fault progression data including advanced and catastrophic fault stages.

Why Public Datasets Matter:
Collecting bearing fault data on your own machine means:

- Waiting months or years for real faults to develop
- Risking machine damage to capture advanced fault data
- Limited variety of fault types and severities
- Public datasets give you:
- Thousands of labeled fault examples – immediately available
- Multiple fault types – inner race, outer race, ball, cage
- Multiple severity levels – from incipient to catastrophic
- Data collected under controlled conditions – clean, well documented labels
- Peer reviewed – used by hundreds of research teams worldwide

The Most Valuable Public Datasets for Rotating Machinery:

Dataset	Institution	Contents	Why valuable
CWRU Bearing Dataset	Case Western Reserve University – USA	Accelerometer data – healthy, inner race, outer race, ball defects – multiple fault sizes and load conditions	Most widely used – benchmark for every bearing fault paper – directly comparable results
MFPT Bearing Dataset	Machinery Failure Prevention Technology – USA	Real bearing fault data – multiple fault types – different loads	Real industrial bearing data – not lab induced
IMS Bearing Dataset	University of Cincinnati – USA	Four bearings run to failure – complete degradation lifecycle from healthy to catastrophic	Complete fault progression – ideal for RUL model training
PRONOSTIA – FEMTO	FEMTO-ST Institute – France	Accelerometer and temperature – bearing run to failure – multiple operating conditions	Multi-sensor – temperature and vibration together – close to your sensor suite
PHM 2012 Challenge	IEEE PHM Society	Bearing degradation – competitive dataset – multiple machines	Well structured – clear evaluation protocol
SEU Gearbox Dataset	Southeast University – China	Gearbox and bearing – multiple fault types	Gearbox specific – useful if machine has gearbox

What Transfer Learning Does:

What Transfer Learning Does:
A model trained on a public dataset already knows what bearing faults look like — in general. Transfer Learning takes that knowledge and adapts it to your specific machine — with only a small amount of your own data.

Public Dataset — CWRU / IMS / FEMTO
↓
Train base model — learns general bearing fault patterns
↓
Your machine — small labeled dataset — even 20 to 50 examples per fault type
↓
Fine tune base model on your data
↓
Model now knows general fault physics AND your machine specific signature
↓
Quantize → Deploy to MCU

Why Your Machine Signature Matters:

Even with the same bearing type — two machines produce different vibration signatures because of:

Factor	Effect on signature
Machine mounting and foundation	Changes resonance frequencies
Coupling type	Affects transmission of vibration
Adjacent machinery	Background vibration contamination
Sensor mounting location	Changes amplitude and frequency response
Operating speed and load	Shifts fault frequency positions

A model trained only on CWRU data will not perform optimally on your machine. Fine tuning on even a small amount of your own data closes this gap significantly.

How Much of Your Own Data is Needed for Fine Tuning:

Scenario	Your own data needed	Expected performance
Same bearing type as public dataset	20 to 50 examples per fault type	Good — public dataset knowledge transfers well
Different bearing type — similar machine	50 to 100 examples per fault type	Good — general fault physics still transfers
Very different machine — different speed / load	100 to 200 examples per fault type	Moderate — more fine tuning needed
Completely novel fault type not in any dataset	Full training from scratch needed	Public dataset not helpful for this fault

Practical Workflow – Combining Public Data and Field Data:

Month 1 to 2 – Before deployment:
- ├── Download CWRU + IMS + FEMTO datasets
- ├── Train base fault classification model
- ├── Validate on held out public dataset examples
- └── This model knows bearing fault physics – general knowledge

Month 2 to 3 – Early deployment:
- ├── Deploy unsupervised anomaly detector – Phase 1
- ├── Simultaneously collect your machine normal data
- └── Begin accumulating any anomaly events and labels

Month 6 to 12 – First fine tuning:
- ├── Combine public dataset examples with your labeled field data
- ├── Fine tune base model on combined dataset
- ├── Validate – compare against anomaly detector performance
- ├── Quantize and redeploy if improvement confirmed
- └── Phase 2 fault classifier now running – informed by public data AND field data

Ongoing:
- ├── Every new labeled fault example added to training set
- ├── Periodic retraining – model improves continuously
- └── Public dataset remains in training mix – never discarded

One Important Caution:

Public datasets are valuable — but they are not a substitute for your own field data. They are a starting point and a supplement — not a replacement.

What public datasets give you	What only your field data gives you
General bearing fault physics	Your specific machine signature
Labeled fault progression data	Your specific operating conditions
Advanced and catastrophic fault examples	Your specific installation effects
A trained base model — immediately	A fine-tuned model — over time
Benchmark to validate your approach	Ground truth for your deployment

You do not have to start from zero. Decades of research have produced high quality publicly available fault data. Use it. Stand on those shoulders — and then add what only your machine can teach.

The Complete Roadmap

From First Deployment to Full Fault Classification – The Complete Roadmap

- Building an ML based rotating machinery health monitoring system is not a single project with a single delivery date.
- It is an engineering program that matures over time – delivering increasing value at each phase while managing the labeled data challenge honestly and practically.
- Let’s look at the Complete Three Phase Roadmap..

PHASE 1 — Foundation Target:
Month 1 to 3 from project start

Activity	Detail
Sensor installation and validation	RTD — front bearing, rear bearing, body. Accelerometer — bearing housing. Acoustic emission — bearing housing. Validate all channels — sampling rates, signal quality, noise floor
Data logging infrastructure	Continuous logging to local storage or edge server. Feature extraction running in real time — 14 feature vector per measurement cycle
Normal data collection	Minimum 4 to 6 weeks — all load conditions, all shifts, all ambient temperatures. Include startups, shutdowns, load transients
Download public datasets	CWRU, IMS, FEMTO — downloaded, preprocessed, feature vectors extracted to match your 14 feature format
Train anomaly detection model	Autoencoder — trained on your normal data. Validate reconstruction error threshold at 99th percentile
Train base fault classifier	Trained on public dataset — general bearing fault knowledge. Not yet deployed — held in reserve
Quantize and deploy anomaly detector	INT8 quantization. Deploy to STM32 via OTA, JTAG, DFU, or supported mechanism
Value delivered	**24/7 autonomous anomaly detection. Early warning of any deviation from normal. Continuous labeled data accumulation begins**

PHASE 2 — Fault Classification Target:
Month 6 to 18 — depending on fault event frequency and fleet size

Activity	Detail
Labeled data accumulation	Every anomaly investigation recorded. Every maintenance event labeled. Every fault confirmation stored with feature vectors
Transfer learning fine tuning	Combine public dataset with accumulated field labeled data. Fine tune base fault classifier on combined dataset
Validation	Compare fault classifier performance against anomaly detector. Confirm improvement before redeployment
Quantize and redeploy	Updated model replacing anomaly detector. Fault classifier now running on MCU
Threshold for redeployment	Minimum 50 confirmed labeled examples per fault type before attempting fault classification
Ongoing retraining cycle	Every 3 to 6 months — retrain on latest accumulated data. Redeploy improved model
Value delivered	**Specific fault identification. Maintenance team knows exactly what to fix before opening the machine. Reduced unnecessary maintenance interventions**

PHASE 3 — Remaining Useful Life Estimation Target: Month 18 to 48 — depending on fleet size and fault event frequency

Activity	Detail
Fault progression data accumulation	Feature vectors labeled with fault severity at each inspection. Complete degradation sequences from healthy to confirmed failure
RUL model training	Supervised regression — input is current feature vector and fault severity trend. Output is estimated time to maintenance intervention
Validation	Compare RUL estimates against actual time to failure from historical events
Deploy RUL model	Replaces or augments fault classifier on MCU. Same hardware — updated model weights
Confidence interval reporting	RUL estimate always reported with uncertainty range — not a single number
Value delivered	**Planned maintenance at exactly the right time. Maximum machine utilisation. Zero unplanned failures on monitored machines**

What Runs on the STM32 MCU at Each Phase:

Component	Phase 1	Phase 2	Phase 3
Feature extraction	Running	Running	Running
FFT computation	Running	Running	Running
Anomaly detection model	Running	Replaced	Replaced
Fault classification model	Not present	Running	Running
RUL estimation model	Not present	Not present	Running
Inference engine — CMSIS-NN	Same throughout	Same throughout	Same throughout
Model weights in flash	Phase 1 weights	Phase 2 weights	Phase 3 weights
Update mechanism	OTA / JTAG / DFU	OTA / JTAG / DFU	OTA / JTAG / DFU

What the FFT Based Approach Does at Each Phase:

The traditional FFT approach is never discarded. It runs in parallel throughout all three phases:

Phase	Role of FFT
Phase 1	Primary diagnosis tool when ML flags anomaly — engineer uses FFT to investigate
Phase 2	Validation layer — confirms ML fault classification with physical frequency evidence
Phase 3	Cross check — FFT severity trend validates RUL estimate from ML

FFT and ML are complementary throughout the entire program lifetime. FFT provides physical grounding and explainability. ML provides early detection and pattern recognition beyond human capability.

The Decision Checklist Before Starting:

Question	Answer needed before proceeding
Is the machine critical enough to justify ML investment?	Quantify cost of unplanned failure vs cost of ML program
Can sensors be installed on this machine?	Confirm mounting locations, cabling, sampling rate feasibility
Is data logging infrastructure available?	Edge storage, connectivity, power for logging system
Is maintenance team willing to record every event?	Data discipline is the program's critical dependency
Is there a fleet of identical machines?	Fleet accelerates labeled data accumulation significantly
What is the realistic timeline expectation?	Phase 1 in months. Phase 2 in 1 to 2 years. Phase 3 in 2 to 4 years

The Single Most Important Success Factor:

Not the algorithm. Not the hardware. Not the framework.

Maintenance data discipline — recording every anomaly investigation, every fault confirmation, every maintenance intervention with precision and consistency — is what determines whether this program reaches Phase 2 and Phase 3.

Without labeled data discipline — the system stays at Phase 1 forever.

The Complete Picture in One View:

	Phase 1	Phase 2	Phase 3
Timeline	Month 1 to 3	Month 6 to 18	Month 18 to 48
ML capability	Anomaly detection	Fault classification	RUL estimation
Data needed	Normal only	50+ examples per fault	Full degradation sequences
Source of data	Your machine	Field events + public datasets	Field fault progressions
Value	Early warning	Know what is wrong	Know when to act
Hardware	STM32 MCU	Same	Same
Model update	OTA / JTAG / DFU	OTA / JTAG / DFU	OTA / JTAG / DFU

This roadmap is honest about timelines, honest about data challenges, and honest about what each phase delivers. The engineering value is real — but it is earned through operational discipline over time, not delivered instantly by an algorithm.

This concludes the Case Study — Rotating Machinery Health Monitoring.