Université
de Technologie
Tarbes
Occitanie Pyrénées

From Signals to Decisions:

Artificial Intelligence Methods for Prognostics and Health Management *Systems*

Prof. Khanh T. P. NGUYEN

Outline

1. **Introduction**
2. Traditional maintenance strategies
3. Basic concepts of AI, and PHM
4. From signals to decisions
   - Diagnostics
   - Prognostics

Motivation

- Explosion of the platform Deepwater Horizon in the Golf of Mexico (USA) in April 2010 (source: sciencesetavenir.fr)

- According to the « Network Rail » in United Kingdom, faults and failures in railway transports are responsible of about 14 millions of minutes of delay per year

- Losses due to failures in petrochemical industry were estimated to 2 millions of $ per day

➡ Need to monitor – assess – diagnose – anticipate – act

\+ Reliability + Availability + Security - Costs

PHM in Industry

Aerospace and Defense

Energy, Oil and Gas

PHM in Industry

Railway transport

Automobile and Automotive

Prognostics & Health Management (PHM)

Definitions

PHM: *means to predict and protect the integrity of equipment and complex systems, and avoid unanticipated operational problems leading to mission performance deficiencies, degradation, and adverse effects to mission safety.* [CALCE Center]

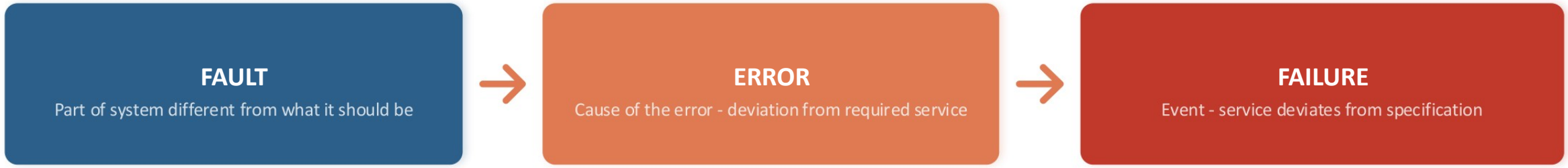
PHM: *discipline that links studies of failure mechanisms to system lifecycle management. PHM uses information to allow early detection of impending or incipient faults, remaining useful life calculations, and logistical decision-making based on predictions* [D. Stark, SEMATECH Inc.]

PHM: *field of research and application which aims at making use of past, present and future information on the environmental, operational and usage conditions of an equipment in order to detect its degradation, diagnose its faults, predict and proactively manage its failures.* [E. Zio, 2012]

Outline

1. Introduction
2. **Traditional maintenance strategies**
3. Basic concepts of AI, and PHM
4. From signals to decisions
    - Diagnostics
    - Prognostics

The Failure Chain - Key Definitions



Three Interconnected Concepts

Fault

A state - the presence of a defect in the system. Can be detected.

Error

A cause - leads to deviation from the required service.

Failure

An event - the service delivered deviates from the service required.

The Cascade Process

Latent Failure → **Effective Failure** → Service interruption. The chain: a **Fault** exists in the system → creates an **Error** (deviation) → leads to a visible **Failure** (event). Understanding this chain is essential for designing effective maintenance and dependability strategies.

What is Maintenance Management?

Definition: Maintenance Management is the **coordination, control, planning, execution, and steering** of activities required to maintain production equipment in operational condition.

Objectives of Maintenance Management

Preserve system functions

Ensure equipment delivers required performance

Avoid failure consequences

Safety risks, production loss, repair costs

Failure Consequences

Safety: Endangerment of people and property

Operational: Production loss and downtime

Non-operational: Repair and replacement costs

Effective maintenance management balances these three risk dimensions through appropriate strategies.

Physical System Management

Characterization of systems: factory, zone, equipment, article, component

Set up intervention tracking systems

Planning & Scheduling

Manage work orders and intervention sequences

Allocate resources: personnel, spare parts, tools

Strategy Determination

Adapt maintenance strategy to context and equipment criticality

Select appropriate approach for each system type

Performance Monitoring

Track KPIs: reliability, availability, costs

Review and challenge strategic choices

Maintenance Strategies Comparison

Strategy	Planned?	Cost Level	Technology	Key Driver	Best For
Emergency	No	Very High	None	Failure event	Critical safety
Scheduled Corrective	Yes	High	Basic	Failure report	Known defects
Preventive Systematic	Yes	Medium	Low	Time/Usage	Wear-prone equip.
Conditional	Yes	Medium	Medium	Degradation signs	Monitored systems
Routine	Yes	Low	Low	Calendar	Running equipment
Predictive	Yes	Variable	High	RUL prediction	Critical assets

Performance Measurement Indicators

"You can only manage what you can measure"

General Indicators

- Budget actual vs planned
- Maintenance costs breakdown
- Overhead, labor, subcontract
- Spare parts costs

Work Order Indicators

- Average processing delay
- Hours per work order
- % corrective vs preventive
- Pending orders count

Dependability KPIs

- Reliability R(t)
- Availability A(t)
- Maintainability M(t)
- MTBF, MTTF, MTTR

Other Important Indicators

Logistics: Spare parts inventory levels, tool availability, infrastructure readiness

Personnel: Subcontractor ratio, production/maintenance operator ratio, overtime proportion, absenteeism rate

Safety/Environment: Incident frequency, environmental impact metrics, compliance scores

Reliability R(t) & Availability A(t)

Reliability R(t)

Probability of **continuous proper functioning** over time.

Physical meaning:

Probability of functioning over a given time without failure.

State model:

E1 (working) --λ--> E2 (failed)

Only considers the working state E1.

Availability A(t)

Probability the system **can execute tasks** when requested.

Physical meaning:

When solicited, the probability that the system works (includes repair).

State model:

E1 (working) ⇔ E2 (failed)

Considers both working and failed states with repair transitions.

Maintainability M(t) & Safety S(t)

Maintainability M(t)

Probability of **locating and repairing** failed elements.

Physical meaning:

Probability of repairing the system within a given time.

State model:

E1 (working) <--μ-- E2 (failed)

Starting from failed state E2, probability of returning to E1.

Safety S(t)

Availability **before catastrophic failure**.

Physical meaning:

Probability of not transitioning to a malignant (catastrophic) failure state.

State model:

E1 (working) <--μ-- E2 (benign failure)

E2 --λ2--> E3 (malignant failure)

Distinguishes benign vs malignant failures.

Average Measures of SDF

MTBF

Mean Time Between Failures

Average duration between consecutive failures of a repaired entity

MTTF

Mean Time To Failure

Average operating time before the first failure

MTTR

Mean Time To Repair

Average duration of repair (pure repair time)

Key Relationships

MTBF = MUT + MDT (Mean Up Time + Mean Down Time)

A = MUT / (MUT + MDT) (Steady-state availability)

Important distinction: MUT ≠ MTTF ≠ MTBF

- **MUT** accounts for imperfect repair (not all components renewed)
- **MTTF** characterizes a fully restored system
- If MDT << MUT, then **MUT ≈ MTBF**

MTTR is pure repair time; MDT includes detection, diagnosis, waiting, and restart times.

FMECA Method

A systematic methodology for analyzing failure modes, effects, and criticality

** This chapter covers the complete AMDEC methodology: initialization, functional decomposition, analysis, and synthesis*

AMDEC - An Overview

AMDEC = **A**nalyse des **M**odes de **D**éfaillance, de leurs **E**ffets et de leur **C**riticité | **FMECA** in English

Industrial Uses

- Predictive reliability analysis
- Equipment reliability optimization
- Maintainability from design stage
- Operational availability mastery

Who Uses AMDEC?

- **Quality** departments
- **Maintenance** departments
- **Engineering** (Bureau d'Etudes)

Often required in specifications and quality standards.

Standard: X 60-510

AMDEC is standardized under **NF X 60-510**. It is a quality method for building or improving quality, requested at the specification level in many industrial contracts.

Types of FMECA

FMECA Product

Analysis of **product design** to improve quality and reliability.

Focus: the product itself, its components, and their interactions.

Applied during the design phase of a new product.

FMECA Process

Analysis of **production operations** to improve manufacturing quality.

Focus: manufacturing steps, process parameters, and controls.

Applied to production lines and manufacturing processes.

FMEAC Procedure

Analysis of **equipment design and operation** to improve availability and safety.

Focus: production equipment, maintenance procedures, and safety systems.

This course focuses primarily on FMEAC Procedure.

AMDEC Objectives

1

Reduce Failures

- Design reliability
- Manufacturing improvement
- Usage optimization
- Preventive maintenance
- Early degradation detection

2

Reduce Downtime

- Maintainability from design
- Testability improvement
- Diagnostic aid
- Corrective maintenance

3

Improve Safety

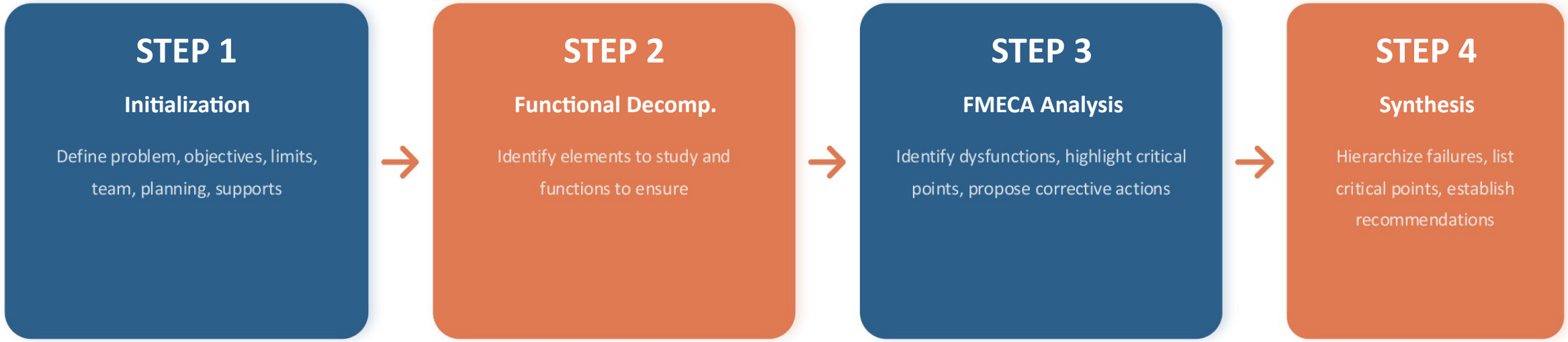
- Identify safety-critical failures
- Risk reduction measures
- Compliance with standards
- Operator protection

Summary: FMEAC is a Quality Method

FMEAC serves as a **construction or improvement tool for quality** It is requested at the specification level and involves cross-functional teams.

The method has **specific steps, standards to respect, and dedicated terminology** It provides a structured framework for identifying, evaluating, and prioritizing risks.

FMECA - The 4-Step Process



Methodology Overview

Step 1: Set the stage - define what to study, why, and with whom

Step 2: Break down the system into functional elements

Step 3: For each element, identify failure modes, causes, effects, and calculate criticality

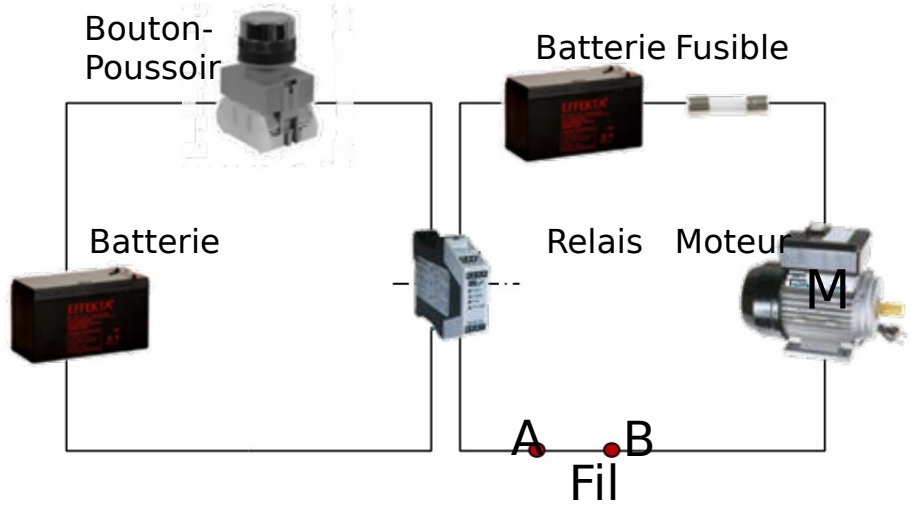
Step 4: Summarize findings and prioritize improvement actions

Each step will be detailed in the following slides.

System Description

Remote control of a DC motor:

- Operator presses push-button (BP) → Relay excited → Contact closes → Motor powered
- When BP released, motor stops
- Fuse protects against short-circuit
- Wire AB passes through zone with **flammable vapors**



Undesirable Event

Overheating of wire AB - risk of fire in flammable vapor zone

Components Analyzed

Push-button (BP), Relay, Fuse, Motor

Assumptions: 1-2 failure modes per component

Operating Assumptions

- System designed for short motor operation time
- Prolonged operation can cause motor destruction via overheating → short-circuit
- After high current from short-circuit, relay contact may remain stuck

Main Function (FP): Generate Motor Energy

Service Functions

FS1: Power Supply (Battery → BP → Relay → Motor)

FS2: Protection (Fuse against short-circuit)

Components & Functions

Component	Function
Push-button (BP)	Allow / interrupt operator command
Relay	Power switching (low → high current)
Fuse	Protect circuit against overcurrent
Motor	Convert electrical to mechanical energy

Scope: Analyze only BP, Relay, Fuse, Motor. Ignore energy sources. Consider 1-2 failure modes per component. The undesirable event is **overheating of wire AB**.

FMEAC Example - Analysis Table

Component	Failure Mode	Cause	Effect	F	G	N
BP	Blocked Off	Foreign body	Loss of power supply	2	2	2
BP	Blocked On	Foreign body / Operator	Prolonged motor operation	2	4	3
Relay	Stuck On	Foreign body / Motor SC	Prolonged operation	2	4	3
Relay	Stuck Off	Foreign body	Loss of power supply	2	2	2
Fuse	Fails to blow	Wrong sizing	Loss of protection	1	5	2
Fuse	Blows unnecessarily	Wrong sizing	Unwanted stop	2	2	1
Motor	Short-circuit	Overheating	High current → wire overheat	2	5	3
Motor	Does not run	Loss of power	No motor energy	2	2	1

FMEAC Limitations & Complementary Tools

AMDEC Limitations

Cannot identify **all events** leading to a specific undesirable event

Cannot determine **precedence order** of failure modes

A **preliminary analysis** that must be completed by other tools

Next: Fault Tree Analysis

For complex systems, FMEAC needs to be complemented by:

- **Fault Tree Analysis (FTA)**

  Top-down deductive approach starting from an undesirable event

- **Other complementary methods:**
  - Reliability Block Diagrams
  - Markov Chains
  - Bayesian Networks

Fault Tree Analysis (FTA)

A deductive method for analyzing how combinations of events lead to system failures

** This chapter covers FTA principles, symbols, gates, construction methodology, and a worked example*

Fault Tree Analysis - Introduction

FTA (Fault Tree Analysis) = A **deductive** analysis method that starts from a single undesirable event and graphically represents the combinations of events leading to its realization.

Core Principles

- **Logical representation** of interactions between events
- **Top-down analysis** by successive levels
- **Search for what should NOT happen** (root event)
- Uses **Boolean logic** (AND, OR gates)

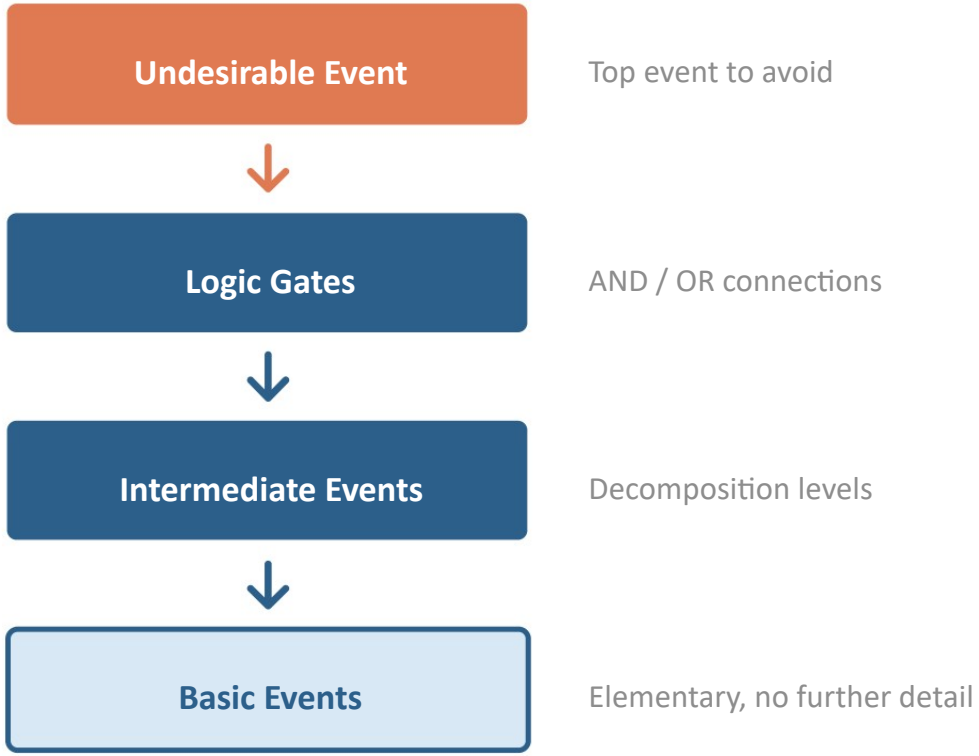
FTA vs AMDEC

AMDEC: Inductive (bottom-up) - starts from components

FTA: Deductive (top-down) - starts from system event

Both are complementary analysis methods

FTA Analysis Flow



FTA - Event Symbols

Rectangle

Resulting Event (Top/Intermediate)

An event resulting from the combination of more elementary events acting through logic gates. Requires further decomposition.

Circle

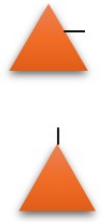
Basic Event (Elementary)

An elementary event that does not need to be developed further. It is the lowest level of the tree. Probability can be assigned.

Diamond

Undeveloped Event

An event that is not considered elementary, but whose causes will not be developed further. Insufficient information or consequence.



Triangle

Transfer Symbol

Indicates that a part of the tree is transferred to another location (in or out). Used for large trees.

Improves readability.

AND Gate

Intersection - All inputs must occur

The output event occurs if **ALL** input events occur simultaneously.

Boolean: $O = A \cap B$

Probability: $P(O) = P(A) \times P(B)$

Example: Motor overheats ONLY IF high current AND cooling failure both occur.

Ask: "What combination of events is needed?"

AND gates reduce the probability of the output event (redundancy effect).

OR Gate

Union - At least one input must occur

The output event occurs if **AT LEAST ONE** input event occurs.

Boolean: $O = A \cup B$

Probability: $P(O) = P(A) + P(B) - P(A \cap B)$

Example: Pump fails IF mechanical failure OR electrical failure OR blockage.

Ask: "What single events can cause this?"

OR gates increase the probability of the output event.

Other gates: INHIBIT (conditional), PRIORITY AND (sequence required), XOR (exactly one) - used for specific logic requirements

1 **Select the Undesirable Event (Top Event)**

Define the system-level event. Must be specific and observable.

2 **Develop Successive Levels**

Each event generated by lower-level events using logic gates.

3 **Stop at Elementary Events**

Stop at elementary events: no further detail needed.

4 **Analyze the Tree**

Identify minimal cut sets and calculate top event probability.

Key Rules for FTA Construction

- **One event at a time**

  Decompose one event before moving to the next

- **Complete the level**

  Finish all events at one level before going deeper

- **Use appropriate gates**

  AND for combined causes, OR for alternative causes

- **Verify logic**

  Check that the tree correctly represents system behavior

- **Document assumptions**

  Record all assumptions and simplifications made

FTA Example - Problem Definition

We reuse the **motor control circuit** example from the AMDEC analysis. The Preliminary Hazard Analysis identified the undesirable event to analyze.

TOP EVENT TO ANALYZE

Overheating of Wire AB (passing through zone with flammable vapors)

Necessary Conditions

- **Condition 1:** Presence of high current in the second circuit
- **Condition 2:** The second circuit remains closed

Logic: Both conditions required = AND gate at top level

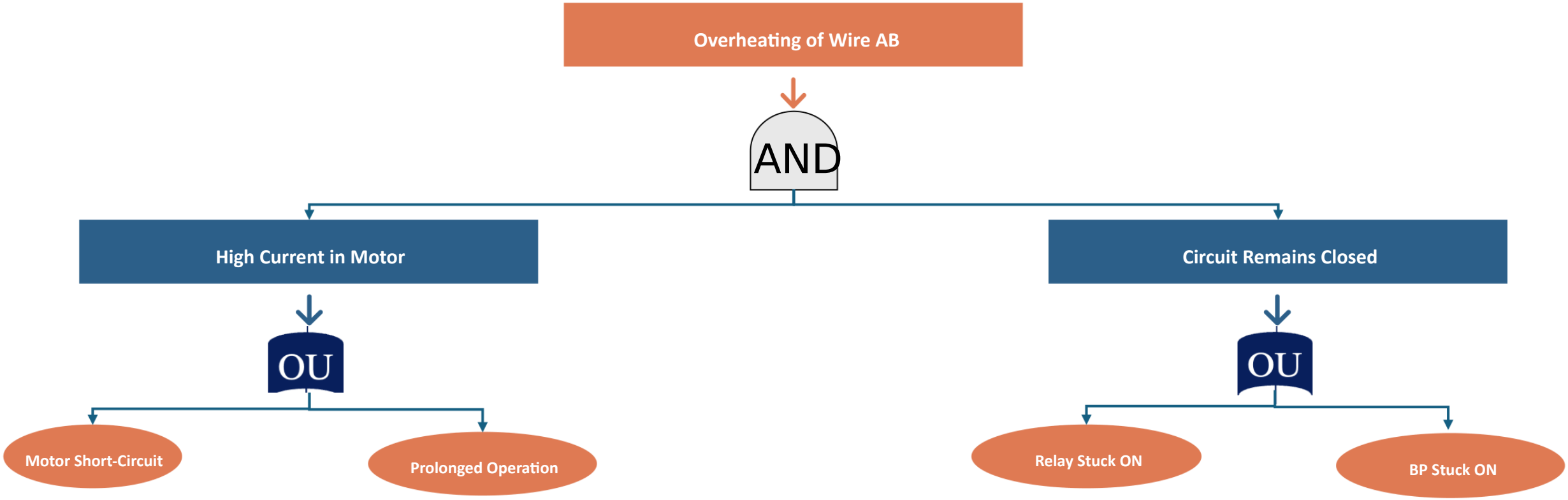
System Assumptions

- System designed for **short motor operation**
- Prolonged operation can cause **motor destruction via overheating**
- After high current from short-circuit, the **relay contact remains stuck** even after de-excitation

Causality Chain

Motor prolonged operation → Motor overheating → Short-circuit → High current → Relay contact remains stuck → Circuit stays closed → **Wire AB overheating**

FTA Example - Building the Fault Tree



Legend

Intermediate event Basic event

Tree structure: Top event (Wire AB overheating) is caused by [High Current] **AND** [Circuit Closed]. Each branch uses an **OR** gate: High Current = Motor Short-Circuit OR Prolonged Operation; Circuit Closed = Relay Stuck ON OR BP Stuck ON.

The AND gate at the top means BOTH conditions must occur simultaneously for the undesirable event.

Qualitative Analysis

Identifying minimal cut sets

Minimal Cut Set: The smallest combination of basic events that causes the top event.

Order 1 (Single failures):

- Motor short-circuit alone (if relay already stuck)

Order 2 (Double failures):

- Prolonged operation + Relay stuck ON
- Prolonged operation + BP stuck ON
- Motor short-circuit + BP stuck ON

Insight: No single basic event causes the top event directly.

Quantitative Analysis

Probability calculation

Boolean Algebra:

AND = Intersection (∩)

OR = Union (∪)

Probability Rules:

P(AND) = Π P(inputs)

P(OR) = 1 - Π (1-P(input))

Importance Measures:

Identify which components contribute most to system failure probability.

Use: Prioritize redesign on highest-contribution events.

AMDEC vs Fault Tree Analysis - Comparison

Dimension	AMDEC / FMECA	Fault Tree Analysis (FTA)
Approach	Inductive (bottom-up)	Deductive (top-down)
Starting Point	Component-level failure modes	System-level undesirable event
Direction	From parts to system effects	From system event to root causes
Strengths	Systematic, exhaustive component review; identifies all failure modes	Focuses on specific critical events; shows logical combinations
Best Used For	Design review, FMEA, reliability improvement	Safety analysis, critical event investigation
Output	Criticality-ranked failure mode list	Logical fault tree, cut sets, top event probability

Complementary Use

AMDEC identifies component failures (what can go wrong at each component) → **FTA analyzes how they combine** to cause system-level failures (how they lead to the top event). Together they provide complete system dependability analysis.

Outline

1. Introduction
2. Traditional maintenance strategies
3. **Basic concepts of AI, and PHM**
4. From signals to decisions
    - Diagnostics
    - Prognostics

Understanding AI

AI represents the capability of computer systems to perform tasks that normally require human intelligence.

- **Definition:** Systems capable of simulating human cognitive functions such as learning, problem-solving, and decision-making.
- **Umbrella Concept:** AI encompasses Machine Learning, Deep Learning, NLP, and other specialized sub-domains.
- **Typology:** Artificial Narrow Intelligence (ANI) vs Artificial General Intelligence (AGI). Today, our progress focuses on ANI.

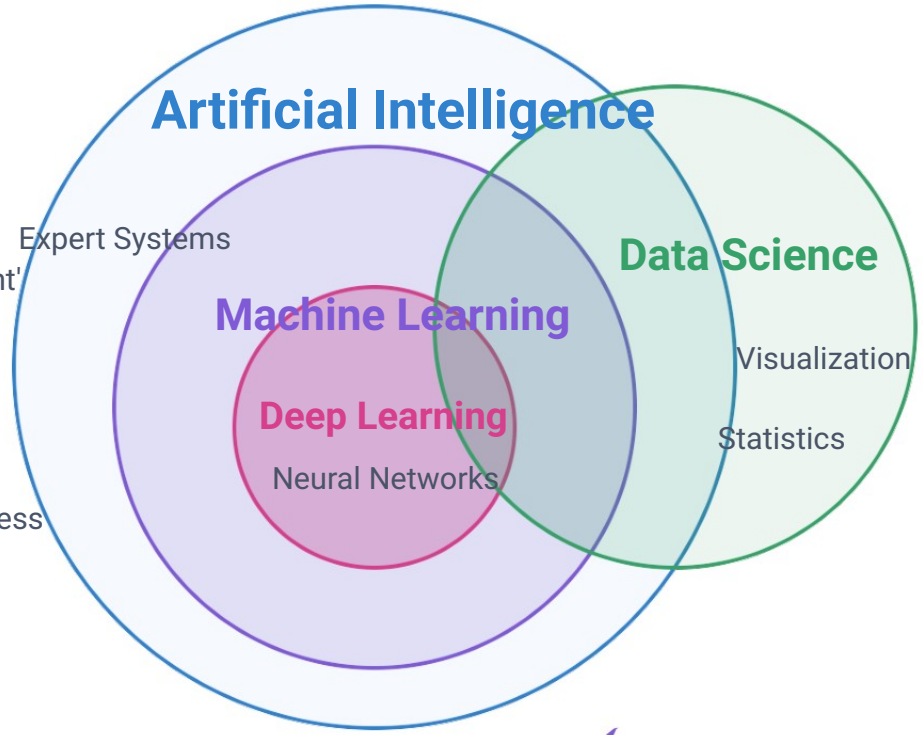
From AI to ML, Data Science & Deep Learning

Artificial Intelligence

Broad field aiming to create systems capable of 'intelligent' tasks: reasoning, learning, adapting.

Deep Learning

Subset of ML using multi-layered neural networks to process complex data.



Machine Learning

Subfield of AI enabling systems to automatically learn from data without expli

Data Science

Analysis and extraction of knowledge from data, using statistics, ML and othe techniques to generate decision-making insights.

Supervised Learning: Learning from labelled data

The algorithm learns to associate inputs (A) with outputs (B) from pairs of examples provided bya "teacher"

Labeled data Each training example includes both the inputs and the expected correct outputs

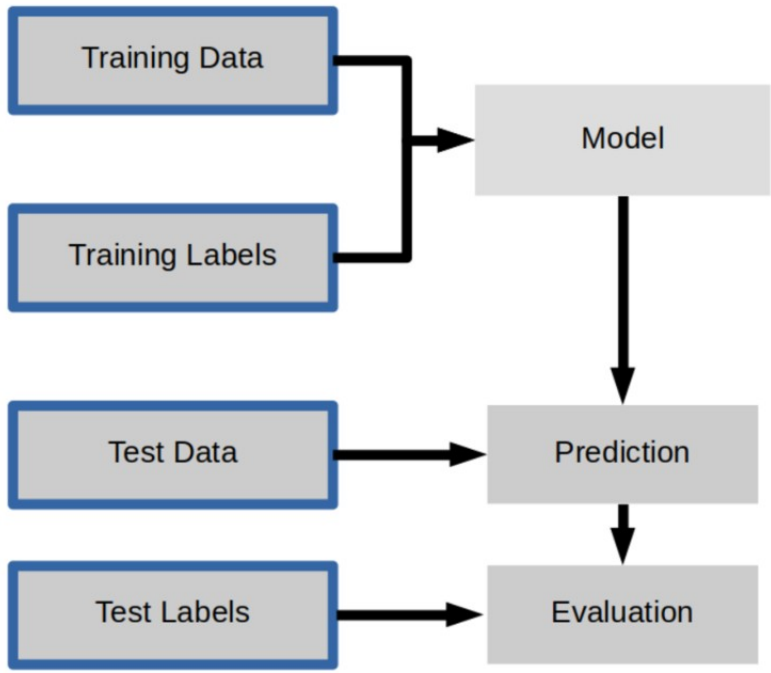
Application examples

Email→Spam/Not Spam Image→Cat/Dog House→Price Audio→Text

Main tasks

Classificationassigns inputs to predefined categories based on patterns learned from labeled data.

Regressionis used when the output is a continuous numerical value rather than a category or class.



Ref. scikit-learn tutorial

Supervisedlearningis the most widely usedformof MLin industry andthe easiest to understand conceptually.

Classification Performance Metrics

Accuracy

- Ratio of correctly classified instances to total instances.
- Accuracy = (TP + TN) / (TP + TN + FP + FN)

Precision

- When model predicts positive, how often is it correct?
- Precision = TP / (TP + FP)

Recall (Sensitivity)

- From all actual positives, how many were identified correctly?
- Recall = TP / (TP + FN)

F1-Score

- Harmonic mean of precision and recall.
- F1 = 2 × (Precision × Recall) / (Precision + Recall)

Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

True Positive (TP): Correctly predicted positive (e.g., fault detected when present)
False Positive (FP): Incorrectly predicted positive (e.g., false alarm)
False Negative (FN): Incorrectly predicted negative (e.g., missed fault)
True Negative (TN): Correctly predicted negative (e.g., healthy identified as healthy)

Confusion Matrix Example

Binary Classification Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	85 True Positive (TP) Correctly predicted as positive	15 False Negative (FN) Incorrectly predicted as negative
Actual Negative	10 False Positive (FP) Model aIncorrectly predicted as positive	90 True Negative (TN) Correctly predicted as negative

Performance Metrics

Accuracy

(TP+TN)/ (TP+TN+FP+FN)

87.5% = (85 + 90) / 200

Precision

TP/ (TP+FP)

89.5% = 85 / (85 + 10)

Recall (Sensitivity)

TP/ (TP+FN)

85.0% = 85 / (85 + 15)

Specificity

TN/ (TN+FP)

90.0% = 90 / (90 + 10)

F1 Score

2 (Precision * Recall)/(Precision +Recall)*

87.2%

ROC Curve & AUC

What is ROC Curve?

Receiver Operating Characteristic curve plots True Positive Rate (sensitivity) against False Positive Rate (1-specificity) at various threshold settings. Shows the trade-off between sensitivity and specificity.

Area Under Curve (AUC)

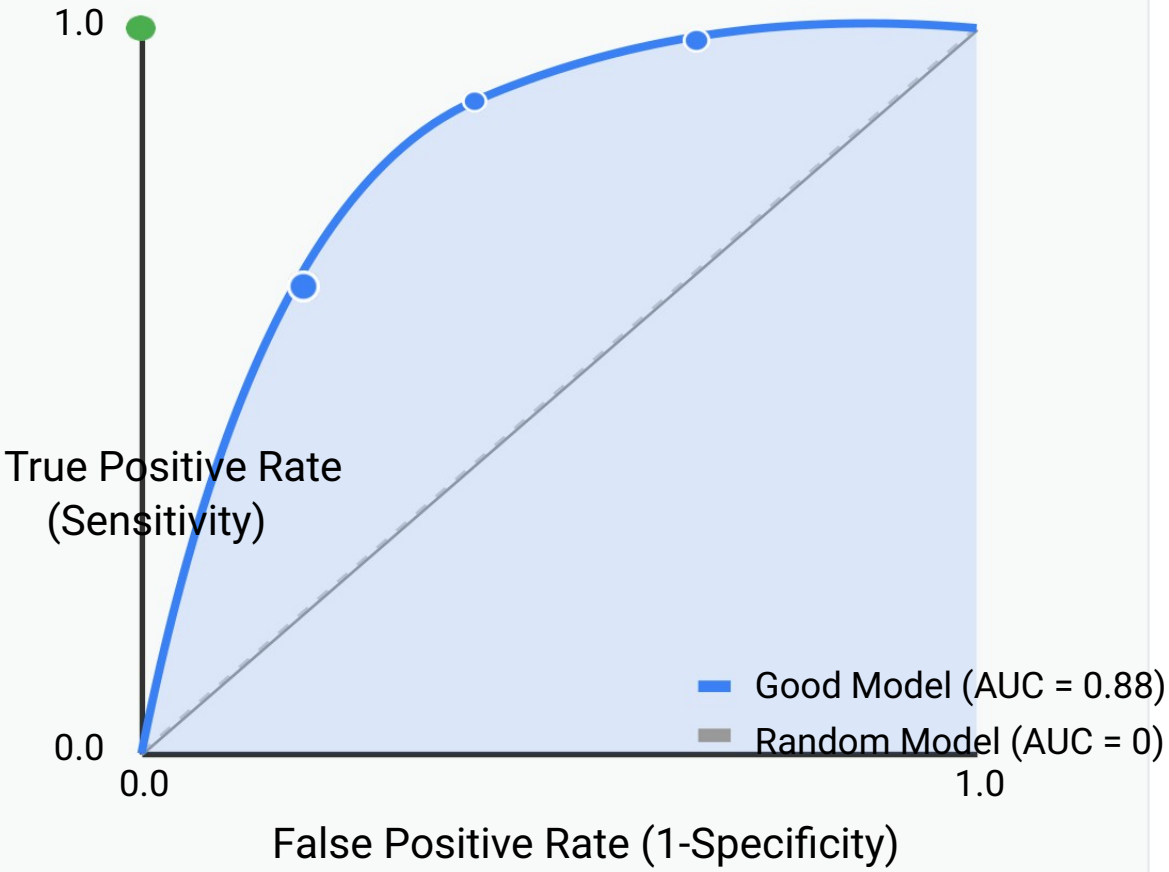
AUC measures the entire two-dimensional area underneath the ROC curve. It represents the probability that a classifier ranks a randomly chosen positive instance higher than a randomly chosen negative one.

Interpreting AUC Values

- AUC = 0.5: No discrimination (random classifier)
- 0.7-0.8: Acceptable discrimination
- 0.8-0.9: Excellent discrimination
- >0.9: Outstanding discrimination

Whento Use ROC-AUC

Best for balanced datasets and when both false positives and false negatives have similar importance. Useful for comparing different classification models, especially with different threshold settings.



ROC Curve showing a model with good discrimination ability
Larger area under curve indicates better modelperformance

Regression Performance Metrics

RMSE (Root Mean Squared Error)

- Measures the square root of the average squared differences between predicted and actual values. Penalizes larger errors more heavily.
- RMSE = √(1/n × Σ(y_pred - y_actual)²)

MAE (Mean Absolute Error)

- Calculates the average of absolute differences between predicted and actual values. More robust to outliers than RMSE.
- MAE = 1/n × Σ|y_pred - y_actual|

R² (Coefficient of Determination)

- Represents the proportion of variance in the dependent variable explained by the model. Values range from 0 to 1 (perfect fit).
- $R^2 = 1 - (\Sigma(y_{actual} - y_{pred})^2 / \Sigma(y_{actual} - y_{mean})^2)$

What is Unsupervised Learning?

Definition

Unsupervised learning is a type of machine learning where algorithms identify patterns in **unlabeled data** without explicit guidance.

Key Principles

- **No labeled training data** - The algorithm works with raw observations without target values
- **Pattern discovery** - Finds hidden structure and relationships within the data
- **Self-organization** - Data points are grouped or mapped based on inherent similarities
- **Exploratory analysis** - Useful when you don't know what patterns to look for

> **Key insight:** The algorithm learns from data structure without being told "right" answers

Unsupervised vs. Supervised Learning

Aspect	Supervised Learning	Unsupervised Learning
Data	Labeled data with input-output pairs Ground truth available for training Often requires manual annotation	Unlabeled data with inputs only No ground truth for direct comparison More abundant and easier to collect
Goals	Predict outcomes for new data Minimize error on known targets Classification or regression tasks	Discover hidden patterns or structures Group similar data points Reduce data complexity/dimensions
Algorithms	Linear & Logistic Regression Decision Trees & Random Forests Support Vector Machines Neural Networks (CNN, RNN) K-Nearest Neighbors	K-Means & Hierarchical Clustering DBSCAN & Mean Shift Principal Component Analysis (PCA) t-SNE & UMAP Autoencoders
Applications	Spam detection & email filtering Medical diagnosis & disease prediction Price & demand forecasting	Customer segmentation & market analysis Anomaly & fraud detection Recommendation systems & topic modeling

Clustering Evaluation: Internal Metrics

Metric	Formula/Description	Interpretation	Range & Implementation
Silhouette Score Well-separated, coherent clusters	Measures how similar an object is to its own cluster compared to other clusters $$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$ where a(i) = intra-cluster distance, b(i) = nearest-cluster distance	Higher values indicate that: • Objects are well matched to their clusters • Poorly matched to neighboring clusters • Clear separation between clusters	**Range:** -1 to 1 **Optimal:** Closer to 1 **scikit-learn:** `from sklearn.metrics import silhouette_score`
Davies-Bouldin Index Minimized ratio of intra-cluster to inter-cluster distances	Average similarity ratio of each cluster with its most similar cluster $$DB = \frac{1}{n} \sum \max_{j \neq i} \frac{\sigma_i + \sigma_j}{d(c_i, c_j)}$$ where σ = cluster dispersion, d = centroid distance	Lower values indicate that: • Clusters are compact (low intra-cluster distances) • Centers are far apart (high inter-cluster distances) • Better separation between clusters	**Range:** 0 and above **Optimal:** Closer to 0 **scikit-learn:** `from sklearn.metrics import davies_bouldin_score`
Calinski-Harabasz Dense clusters with clear separation	Ratio of between-cluster variance to within-cluster variance $$CH = \frac{Tr(B_k)/(k-1)}{Tr(W_k)/(n-k)}$$ where B_k = between-group dispersion, W_k = within-group dispersion	Higher values indicate that: • Clusters are dense (low intra-cluster variance) • Well-separated (high inter-cluster variance) • Also known as the Variance Ratio Criterion	**Range:** 0 and above **Optimal:** Higher values **scikit-learn:** `from sklearn.metrics import calinski_harabasz_score`

Clustering Evaluation: External Metrics

External Evaluation

Comparing clustering results with known ground truth labels

Metric Selection

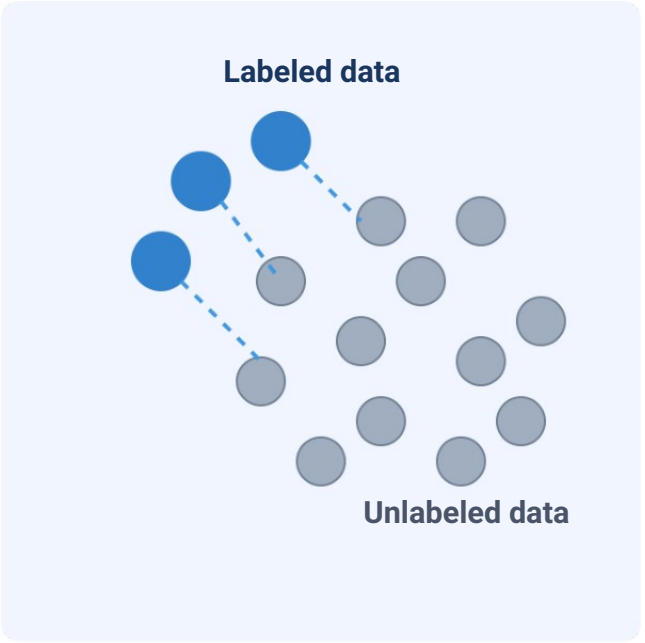
Choose metrics based on your evaluation criteria and use case

Metric	Description	Range & Formula	When to Use
ARI	**Adjusted Rand Index**: Measures similarity between two clusterings, adjusted for chance	Range: [-1, 1] `1.0` is perfect match `0.0` is random labeling	Best when clusters are of similar size and number. Accounts for both precision and recall.
AMI	**Adjusted Mutual Information**: Measures mutual information between two clusterings, adjusted for chance	Range: [0, 1] `1.0` is perfect correlation `0.0` is no correlation	Robust with varying cluster numbers between partitions. Less sensitive to cluster size variations.
Homogeneity	Measures if each cluster contains only members of a single class	Range: [0, 1] `h = 1 - H(C|K)/H(C)` where H is entropy	When you care about cluster purity. Perfect when each cluster has only one class.
Completeness	Measures if all members of a given class are assigned to the same cluster	Range: [0, 1] `c = 1 - H(K|C)/H(K)` where H is entropy	When you want all points of same class to be in the same cluster. Perfect when each class is in a single cluster.
V-measure	Harmonic mean of homogeneity and completeness	Range: [0, 1] `v = 2 * (h * c)/(h + c)` where h, c are homogeneity & completeness	When you need a balanced evaluation of cluster purity and completeness in a single metric.
FMI	**Fowlkes-Mallows Index**: Geometric mean of pairwise precision and recall	Range: [0, 1] `FMI = √(PPV * TPR)` where PPV is precision, TPR is recall	When you want to balance between precision and recall. Helpful for imbalanced datasets.

Semi-Supervised Learning: Hybrid approach

Semi-supervised learning combines a small number of labeled examples with a large amount of unlabeled data.

- **Strategic compromise** Leverages the power of unlabeled data while benefiting from the precision of labeled data
- **Cost-effective solution** Ideal when data labeling is expensive, time-consuming, or requires experts
- **Real-world examples** ImageNet (millions of images, minority annotated), speech recognition, document classification
- **Techniques** Self-training, co-training, similarity graphs between data

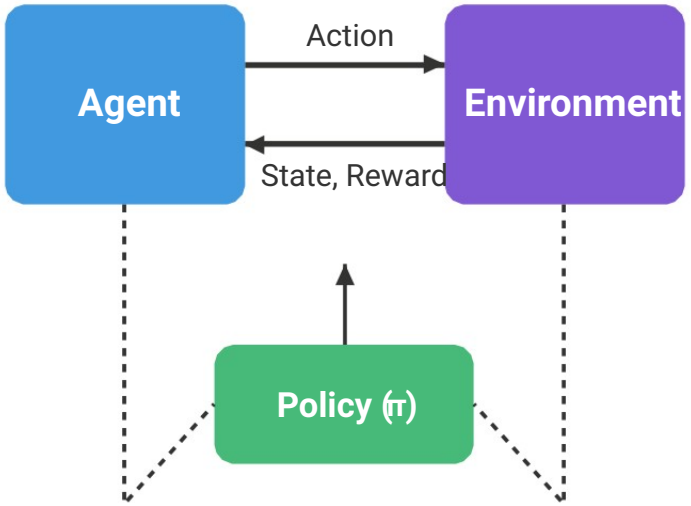


The balancebetweenlabeled andunlabeled data depends on the application domainandthe cost of labeling.

ReinforcementLearning: Learning by Interacting with Environment

An agent learns optimal behavior through trial and error by receiving feedback from its actions.

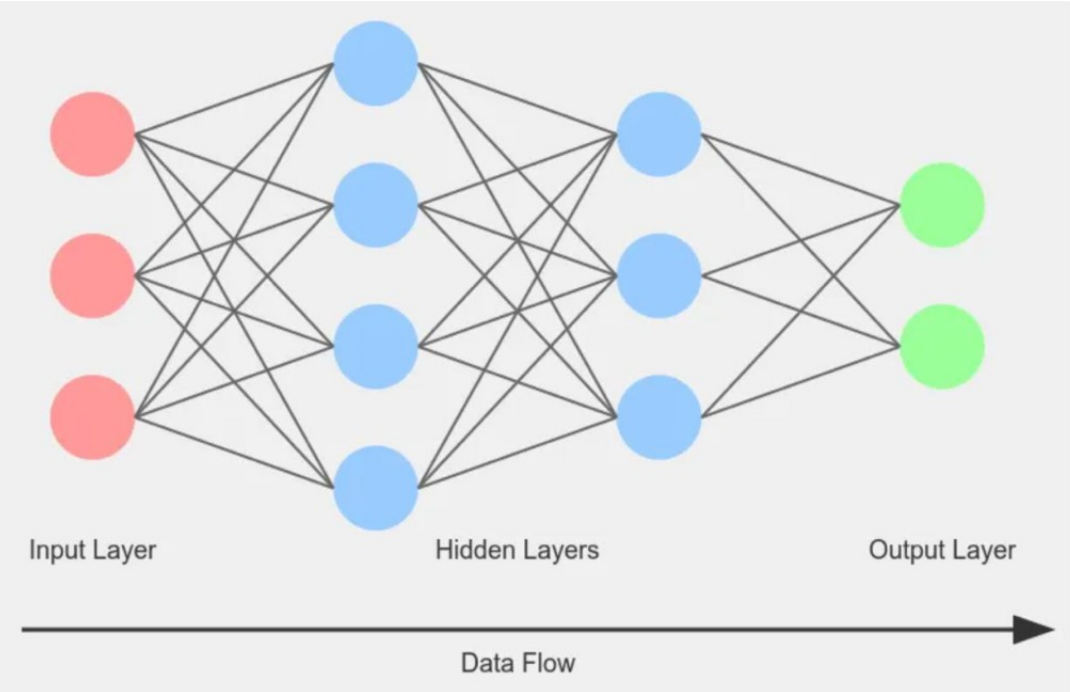
- **Agent-Environment Interaction**: Agent takes actions in an environment and receives rewards/penalties
- **Reward Maximization**: Goal is to learn policy that maximizes cumulative reward over time
- **Real-World Applications**: Game playing AI (AlphaGo), autonomous vehicles, robotics, recommendation systems
- **Analogy**: Similar to training a pet with treats - positive behavior gets rewarded



The reinforcement learningloop

Unlike supervised learning, reinforcement learning doesn't require labeled examples but learns through experience.

DeepLearning&NeuralNetworks



Note: Despite the brain-inspired name, neural networks function very differently from biological brains

Definition

Neural networks with many layers ("deep") that can learn complex patterns and representations from data

ML Relationship

Deep learning is a subset of machine learning that performs exceptionally well with large datasets

Applications

Image recognition, natural language processing, speech recognition, recommendation systems

Parameters vsHyperparameters

Parameters

Internal values learned during training

Definition

Values that the model learns during training from data

Examples

- Weights in neural networks
- Coefficients in linear regression
- Support vectors in SVM
- Tree structure in decision trees

Howset

Automatically optimized during model training

Hyperparameters

Configuration values set before training

Definition

External configuration settings that govern training process

Examples

- Learning rate in gradient descent
- Number of trees in random forest
- Number of layers in neural networks
- Regularization strength (λ)

Howset

Manually or via hyperparameter tuning methods

Key distinction: Parameters are learned from data; hyperparameters are set by the data scientist.

Examples of Hyperparameters

Learning Rate

Controls step size in gradient descent. Toohigh: overshooting optimal solution. Toolow: slow convergence or getting stuck in local minima.

Tree Depth (max_depth)

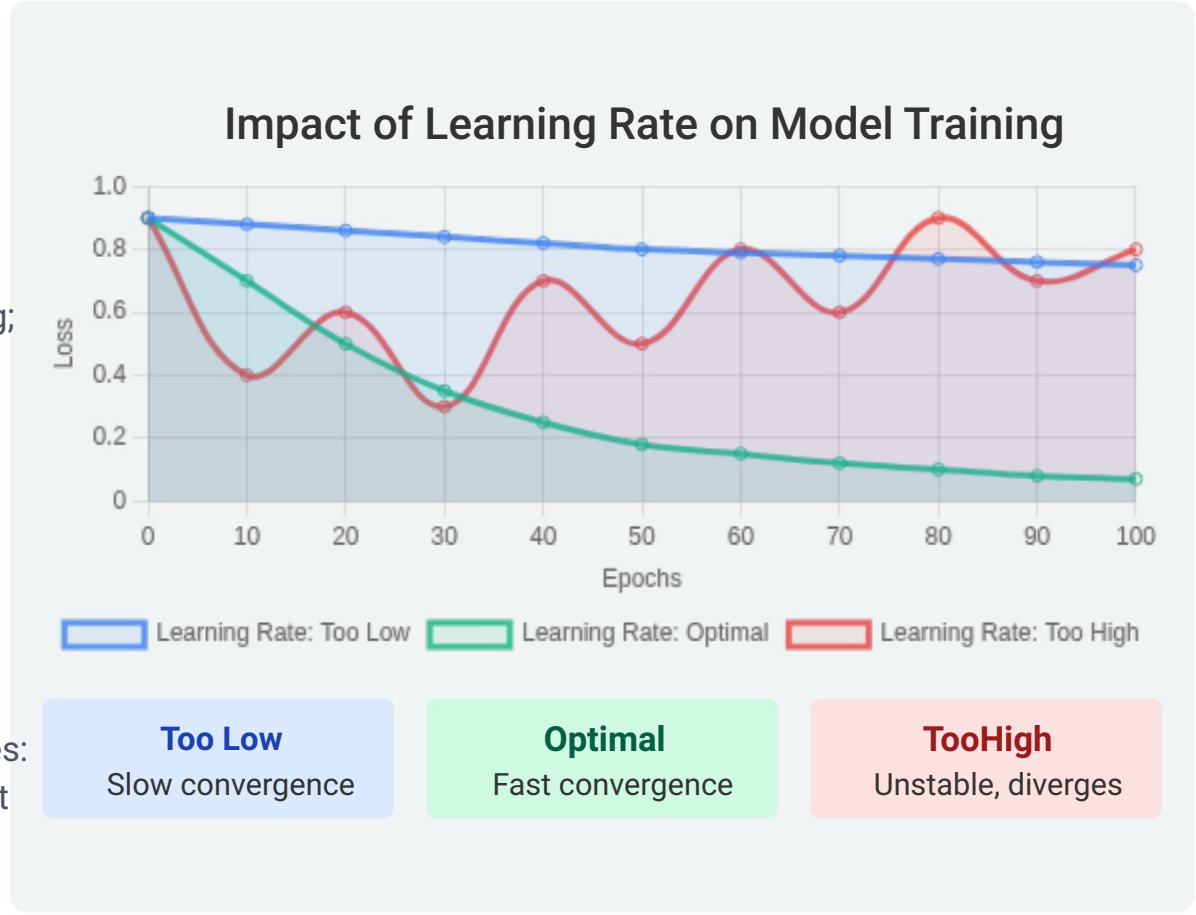
Maximum depth of decision trees. Higher values can lead to overfitting; lower values may cause underfitting. Typical range: 3-10.

Regularization()

Controls model complexity. Higher values promote simpler models to prevent overfitting. L1 (Lasso) and L2 (Ridge) are common types.

Batch Size

Number of samples processed before updating model. Smaller batches: more updates but noisier gradients. Larger batches: stable but may get stuck.



Source: Andrew Ng's Machine Learning Specialization, Coursera; Stanford CS229

Bias-Variance Trade-off

Bias (Underfitting)

Error from overly simplified models that miss relevant relations between features and target outputs. High bias models perform poorly on both training and test data.

Finding the Sweet Spot

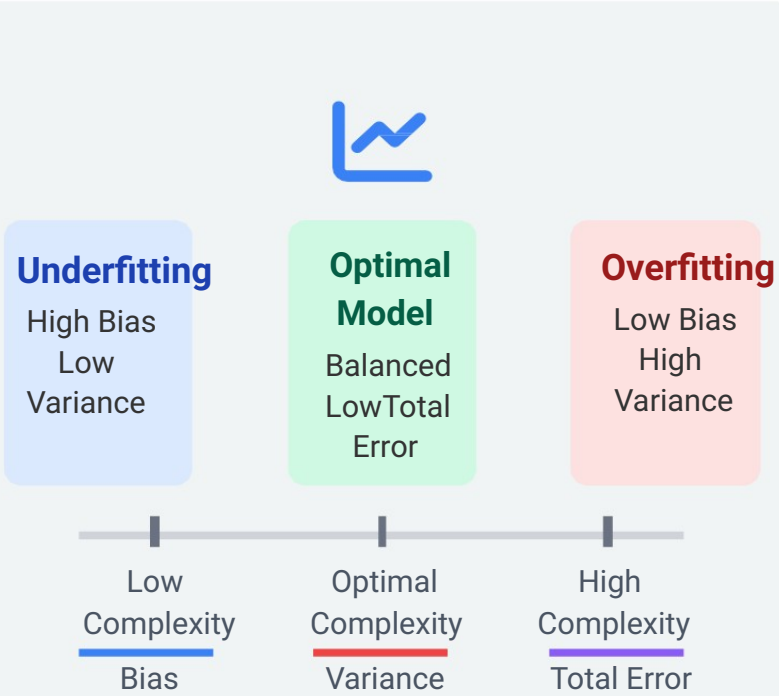
Use techniques like cross-validation to identify the model complexity that achieves the best generalization performance on unseen data.

Variance (Overfitting)

Error from excessive model complexity capturing noise in training data. High variance models perform very well on training data but poorly on unseen test data.

The Trade-off

As model complexity increases, bias decreases but variance increases. The goal is to find the optimal model complexity that minimizes total error (bias² + variance + irreducible error).



Preventing Overfitting

Regularization

L1 (Lasso): Adds |w| penalty, promotes sparsity by forcing some weights to zero

L2 (Ridge): Adds w^2 penalty, shrinks weights toward zero without eliminating features

Cross-Validation

k-fold: Split data into k subsets, train on k-1 and validate on the remaining fold

Helps assess model generalization capability on unseen data

Early Stopping

Stop training when validation error starts increasing while training error continues to decrease

Dropout

Randomly deactivates neurons during training, forcing the network to be more robust

Overfitting
High train accuracy
Poor validation accuracy

Optimal Model
Good train accuracy
Good validation accuracy

With proper regularization:
Reduced variance without significantly increasing bias
Better generalization to unseen data

Python & Public Libraries for AI/ML

TheStandardToolkitfor AI Development

Python has become the de facto standard programming language for AI and ML thanks to its simplicity, readability, and powerful ecosystem of libraries.

Scikit-learn
Traditional ML algorithms, data preprocessing, model evaluation

TensorFlow
End-to-end ML platform for large-scale neural networks and deployment

Keras
High-level neural networks API, focused on user-friendliness

PyTorch
Dynamic neural networks with strong GPU acceleration

These libraries provide pre-built components that simplify implementing complexAI algorithms and models without writing everything from scratch.

Python Libraries for Machine Learning

Essential Python libraries for data processing, analysis, and machine learning. These libraries provide the foundation for efficient ML workflows.

NumPy

Foundation for scientific computing in Python. Provides support for large, multi-dimensional arrays and matrices, along with mathematical functions.

pandas

Data manipulation and analysis library offering DataFrame structures, data cleaning, transformation, and exploration capabilities.

scikit-learn

Machine learning library with algorithms for classification, regression, clustering, and preprocessing. Simple and efficient tools for data analysis.

matplotlib

Visualization library for creating static, interactive, and animated plots. Essential for data exploration and communicating results.

Key Benefits

- Seamless integration between libraries
- Extensive documentation and community support
- High performance with optimized backends
- Support for diverse data formats and operations

Google Colab: Introduction & Demo

Google Colab is a free cloud-based Jupyter notebook environment with GPU/TPU support and Google Drive integration:

```
# Mount Google Drive for data accessfrom
google.colab import drive
drive.mount('/content/drive)

# Load and explore datasetimport
pandas as pd
import matplotlib.pyplotas plt import
seaborn as sns

# Load sample dataset (Colab includes many datasets)
df = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv')

# Quick data exploration
print("Dataset shape:", df.shape) print("\nFirst 3
rows:")
print(df.head(3))

# Create a visualization
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x='sepal_length', y='sepal_width', hue='species') plt.title('Iris Dataset: Sepal
Dimensions by Species')
plt.show()
```

Key Features:

- Free GPU/TPU access for computation
- Pre-installed ML libraries
- Google Drive integration
- Collaborative editing

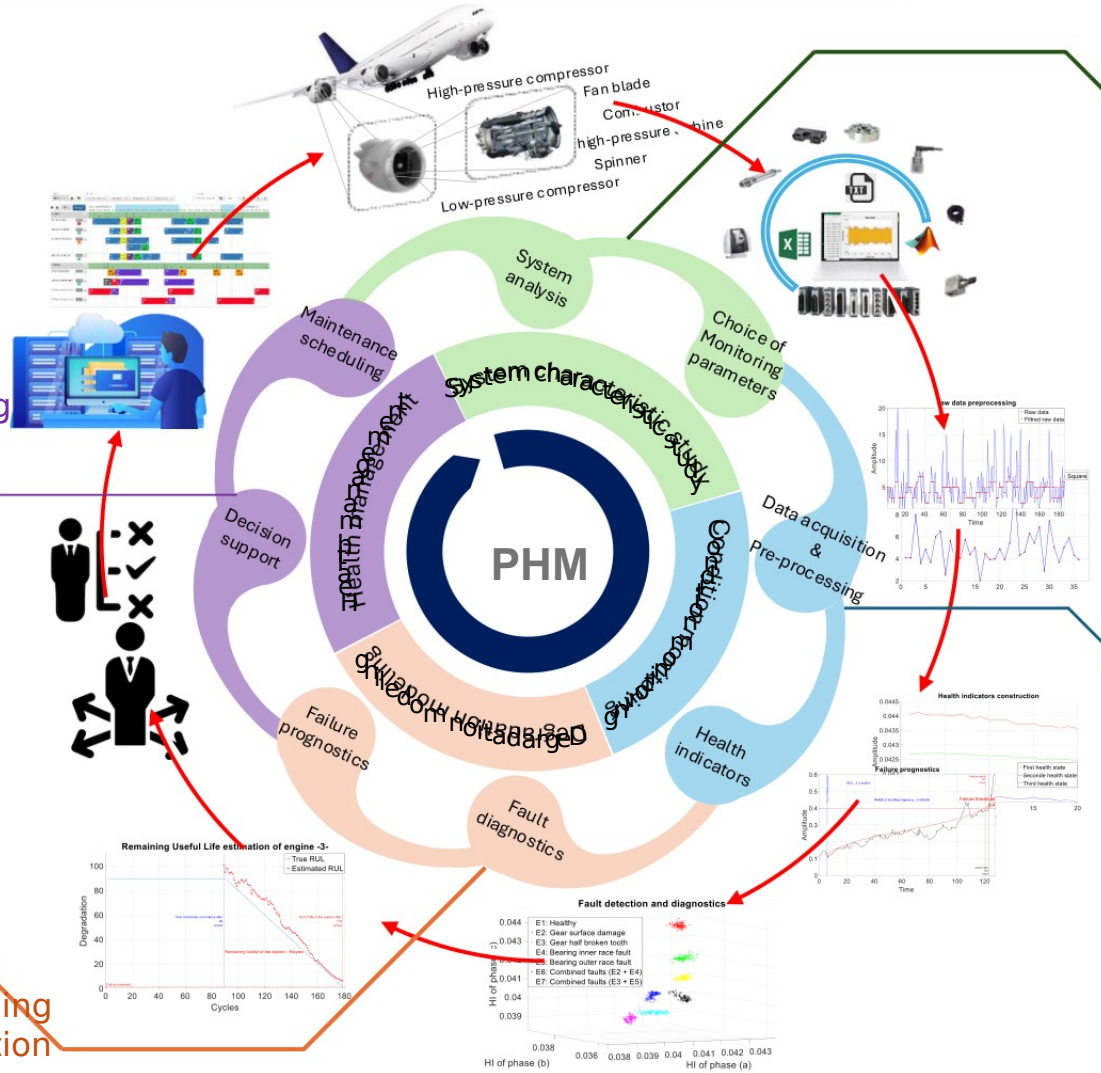
Getting Started:

- Visit colab.research.google.com
- Create a new notebook
- Choose Runtime > Change runtime type for GPU
- Share notebooks with collaborators

Data-driven Prognostics and Health Management

- System architecture study
- System structure behavior study
- Critical component identification
- Physical parameters identification
- Sensors placement

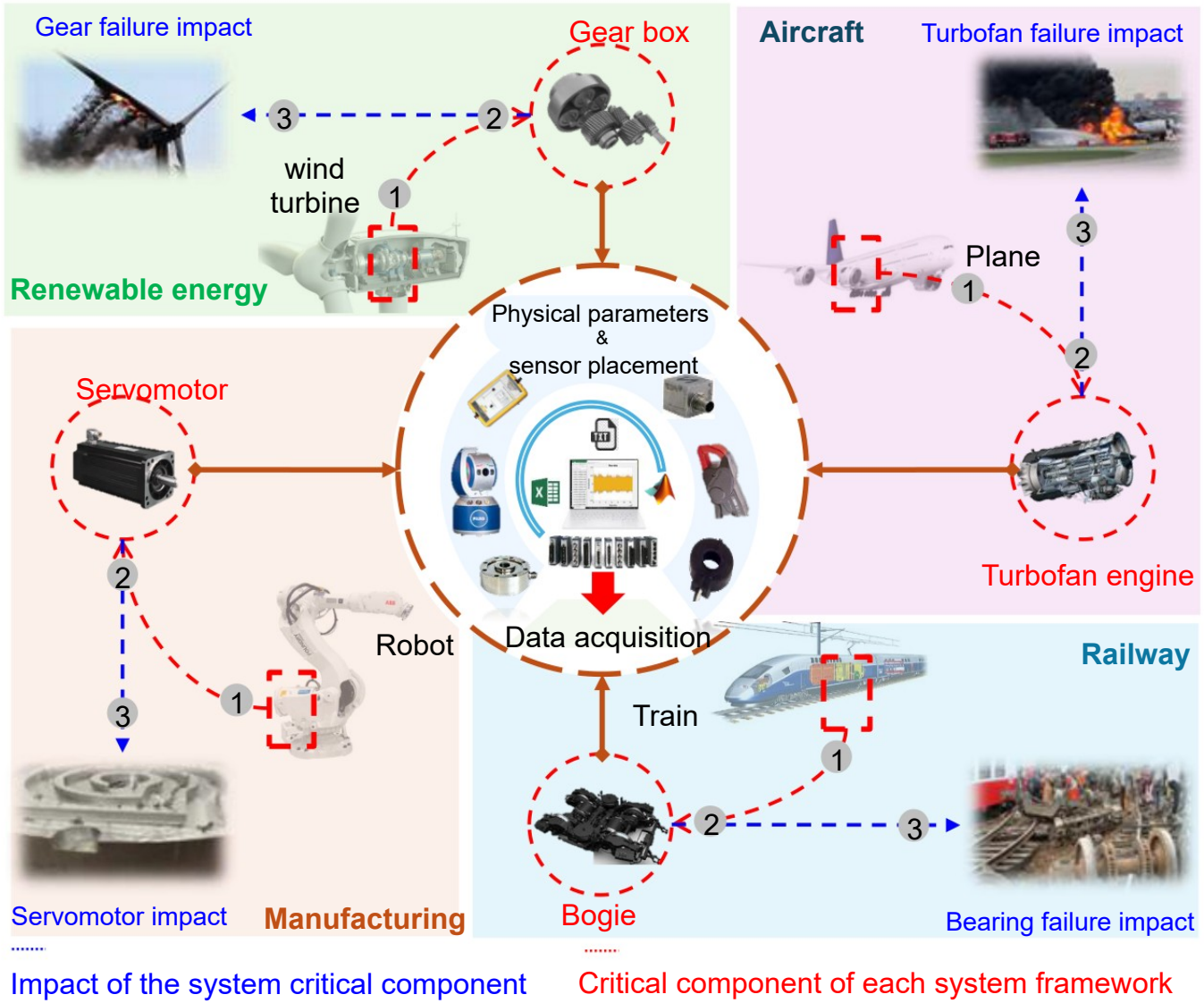
- Alerts generation
- Inventory check
- Maintenance decision making
- Asset maintenance strategy
- Assignment of tasks



- Data acquisition
- Data pre-processing
- Features extraction
- Health indicators construction

- Fault recognition
- Fault identification and localization
- Degradation evolution modeling
- Remaining useful life estimation

System characteristic study

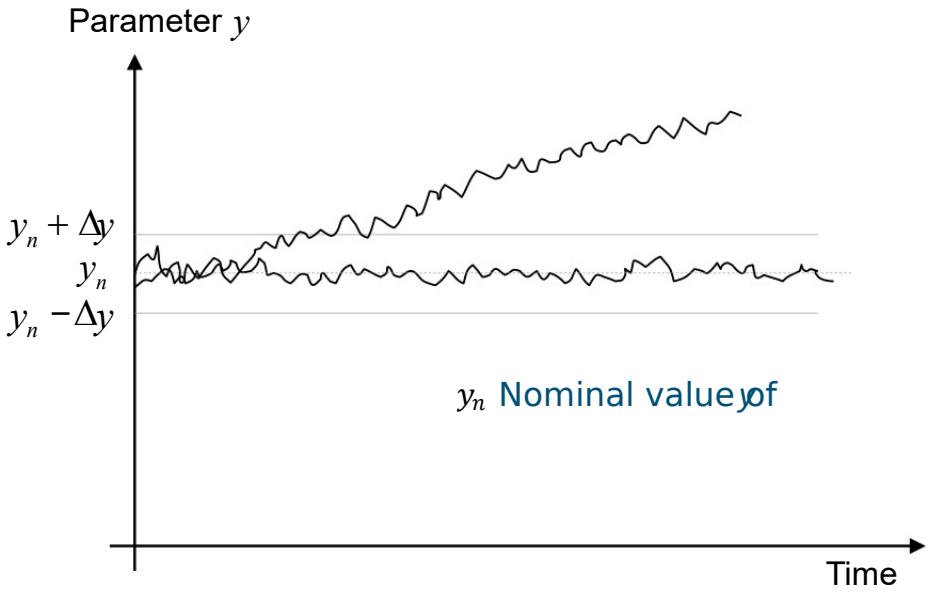


- System architecture study
  - Functionalities
  - Domain application
  - ...
- System structure behavior study
  - Operating conditions
  - Systems interaction
  - ...
- Critical component identification
  - Subsystem
  - Component
  - ....
- Physical parameters identification
  - Electrical
  - Mechanical
  - ...
- Sensors placement
  - Sensor location
  - Sensitivity
  - ...

Condition monitoring & fault detection

Condition monitoring: Acquisition and processing of information and data that indicate the state of a machine over time.

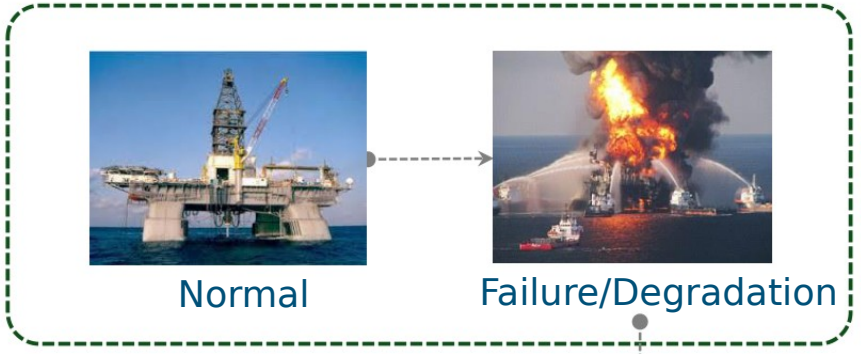
Detection: Fault (or failure) detection consists of deciding whether a system is in its nominal operation (state) or not. [NF E90-372, NF ISO 13372 : 2012-12]

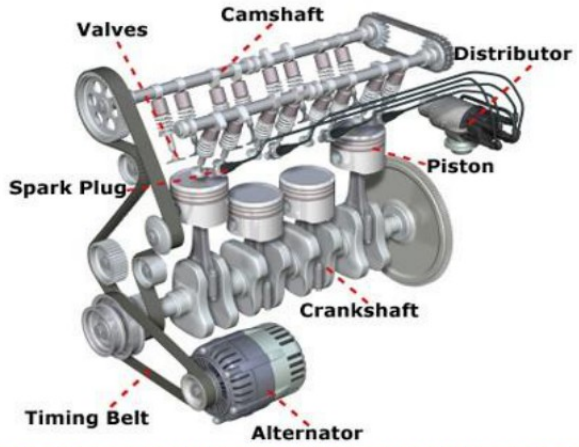


Monitoring or observation of a parameter

Fault diagnostics

Examination of symptoms and syndromes to determine the nature of faults or failures (kind, situation, extent). [NF E90-372, NF ISO 13372 : 2012-12]





1st

Which component is failed or degraded?

2nd

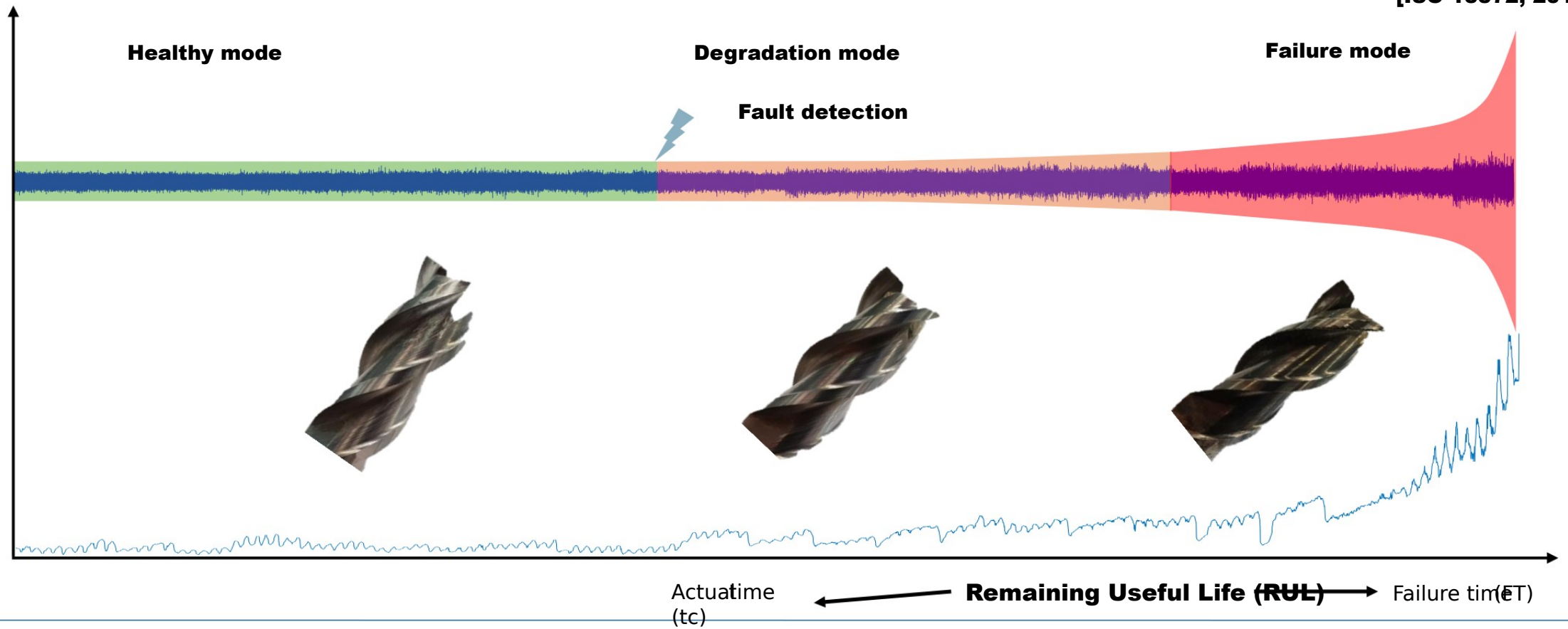
What caused it to fail or degrade?

- Overheating ?
- Lack of lubrication ?
- Detonation ?
- Misassemble ?

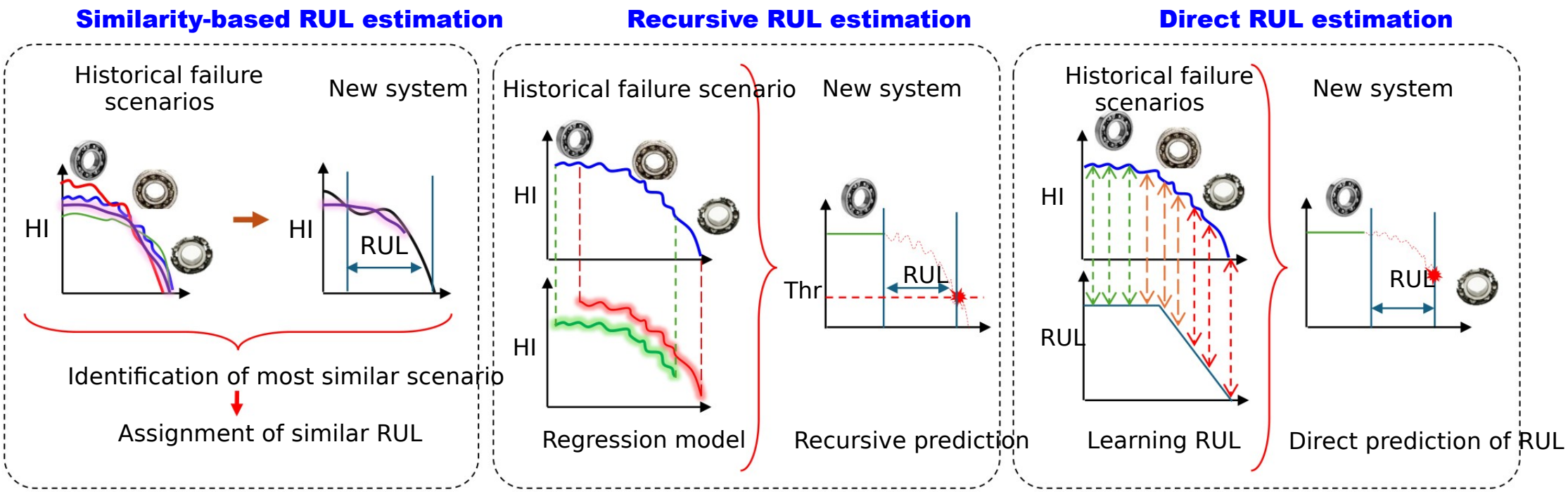
Fault prognostics

Failure prognostic is the analysis of the symptoms of faults to predict future condition and residual life within design

[ISO 13372, 2012 1.5]



Remaining useful life techniques

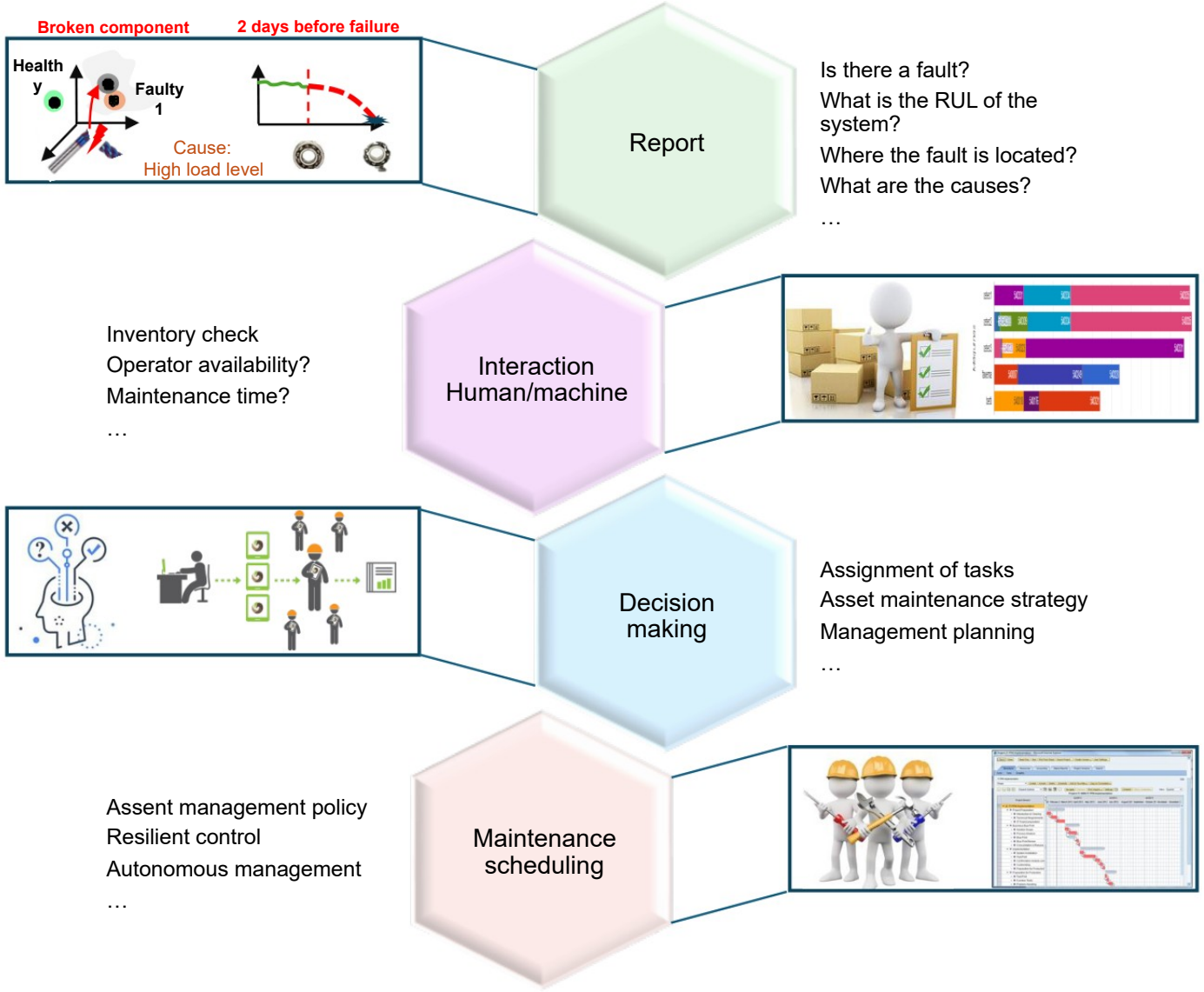


- ✓ Easy implementation and fast computing time
- ✓ Perform good predictions
- ✗ Sensitive to condition variations
- ✗ Require large amount of similar data

- ✓ Easy for threshold identification
- ✓ Appropriate for incomplete data
- ✗ Difficult in non-stationary conditions
- ✗ Reduced modeling quality due to low data

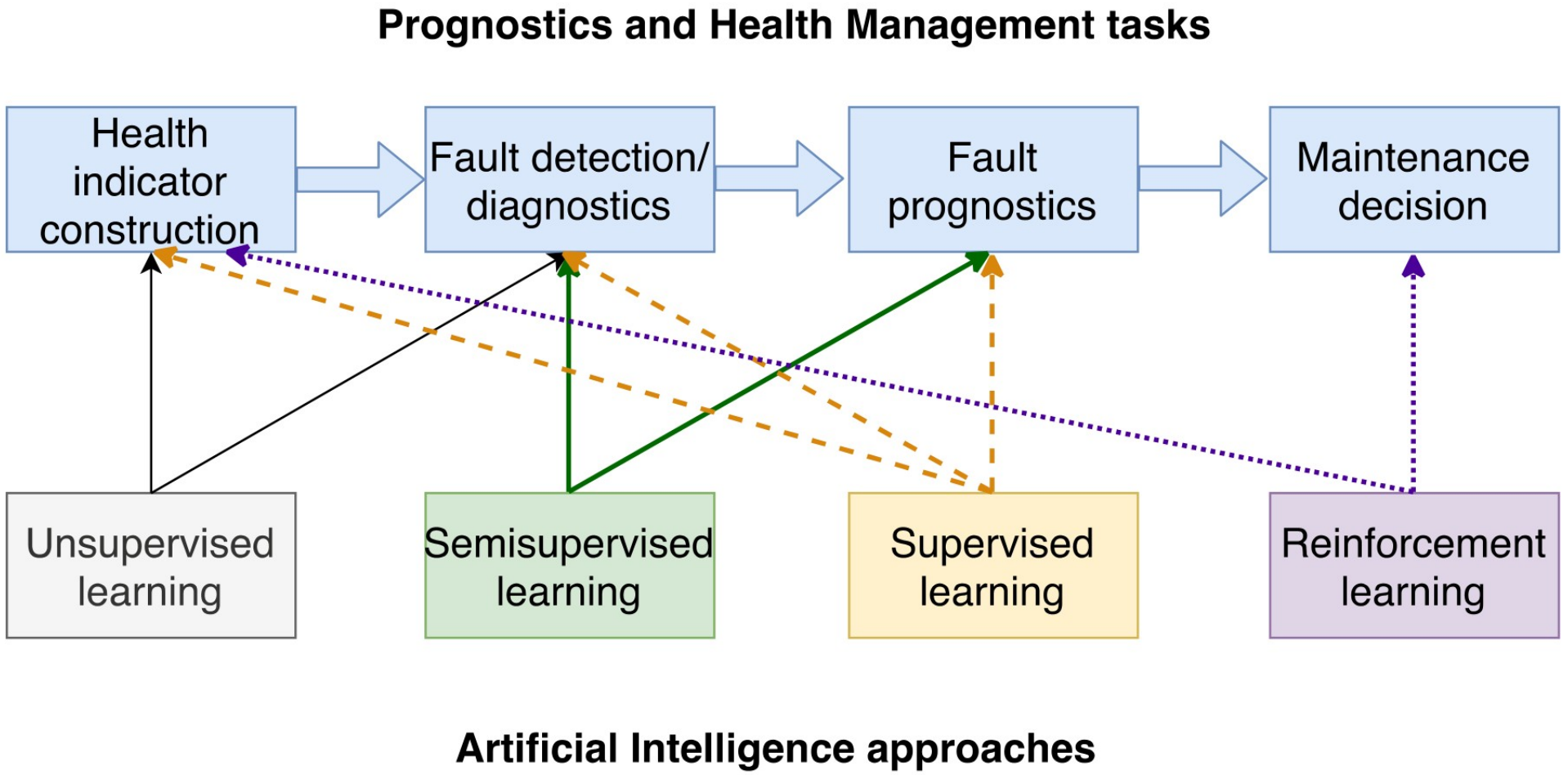
- ✓ Perform high accuracy prediction
- ✓ Adaptive to the system variation
- ✗ Require large amount of similar data
- ✗ Take a lot of computation time

Decision support for system health managem



- Alerts generation
  - Threshold identification
- Inventory check
  - Spare parts forecasting
  - Stock level auditing
  - …
- Maintenance decision making
  - Cost-benefit analysis
  - …
- Asset maintenance strategy
  - Preventive vs. predictive planning
  - Maintenance policy optimization
  - …
- Assignment of tasks
  - Resource allocation
  - Scheduling optimization
  - …

Contributions of AI approaches to PHM tasks



A review of artificial intelligence methods for engineering prognostics and health management with implementation guidelines
Nguyen, K Medjaher, DT Tran, Artificial Intelligence Review 56 (4), 3659-3709

Outline

1. Introduction
2. Traditional maintenance strategies
3. Basic concepts of AI, and PHM
4. **From signals to decisions**
   - **Diagnostics**
   - Prognostics

Case Study - Diagnostics

Key Components

Motor: 3-phase induction, 0.09 kW, 220V, 0.70A

Inverter: WEG CFW300 vector drive for speed control

Transmission: Belt-pulley (A60/A80 pulleys, V-profile belt)

Shaft & Disc: 149 mm disc, 36 holes, 6.5 mm thick

Bearings: Two 1205-type bearings in SN505 housings

Sensors: 4 accelerometers (vertical/horizontal positions)

Four states:

Normal Operation
Baseline condition

Misalignment
Shaft offset introduced

Unbalance
Added mass on rotor disc

Mechanical Looseness
Bearing housing loosened

Data description

Dataset Parameters

Sampling frequency: 25 kHz (25,000 samples/sec)

Signal length: 25,000-time steps per sample

Shaft speed: 1,238 RPM

Full dataset: 8,400 signals per sensor x 4 sensors

Array Shapes

Full data per condition: (8400, 25000)

Tutorial subset: (500, 25000) per condition

Total features after extraction: 255 per sample

Feature matrix: (2000, 255)

Dataset Summary Table

Condition	Samples Used	Signal Length	Sampling Frequency
Normal	500	25,000	25 kHz
Misalignment	500	25,000	25 kHz
Unbalance	500	25,000	25 kHz
Looseness	500	25,000	25 kHz

⚠ **Memory consideration:** The full dataset is 1.91 GB. We load only 500 samples per class to stay within Colab memory limits. The .npz format allows memory-mapped access.

Downloading and Loading Data

CODE

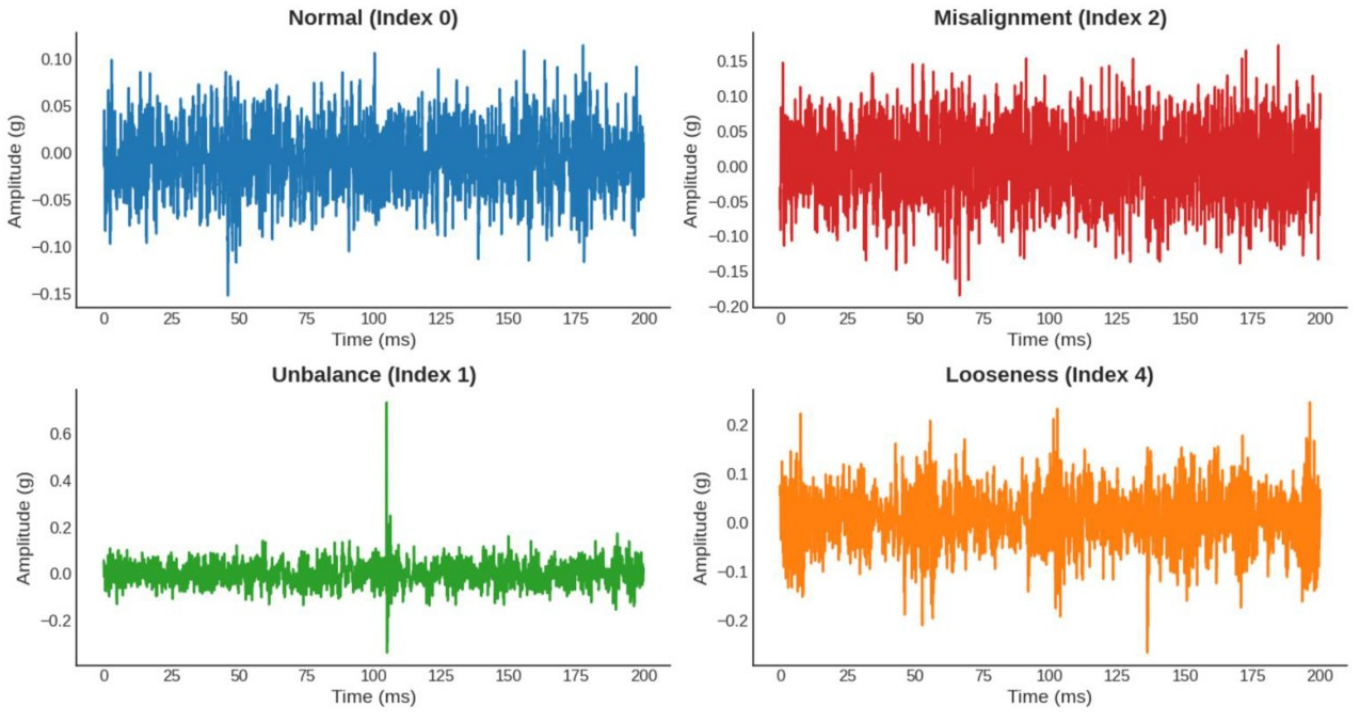
```
import gdown, numpy as np
url = 'https://drive.google.com/uc?id=FILE_ID'
gdown.download(url, output='processed_fault_data.npz')
def load_fault_data(file_path, num_samples=300):
with np.load(file_path) as data:
normal = data['normal'][:num_samples].copy()
# ... load misalignment, unbalance, looseness
return normal, misalign, unbalance, looseness, fs, rpm
```

Code Explanation

Aspect	Description	Details
What	Downloads processed vibration data from Goog le Drive	Uses g down library for Google Drive file access
Why	Need vibration sig nals to train and test the diagnostic model	Real-world data from controlled test bench experiments
Input	Goog le Drive URL, num_samples parameter	Default 300 samples per class for memory efficiency
O utput	4 data arrays + fs (25000 Hz) + rpm (1238)	Each array shape: (num_samples, 25000)

Key point: .npz keys: 'normal', 'misalignment', 'unbalance', 'looseness', 'fs', 'rpm'. Use mmap_mode='r' to inspect without loading.

Raw Signal Visualization — Time Domain



X-Axis: Time (milliseconds)

Duration shown: 0.2 s = 5,000 samples at 25 kHz

kHz

Y-Axis: Acceleration (g)

Amplitude of vibration measured by accelerometer

accelerometer

Condition Comparison

Normal: Lower amplitude, regular pattern

Misalignment: Higher periodic peaks

Unbalance: Strong sinusoidal component

Looseness: Irregular, impulsive behavior

Key message: Time-domain signals show visible differences but are not always sufficiently discriminative for automatic classification. Statistical features and

Statistical features and frequency-domain analysis are needed to make fault patterns more distinguishable.

Data Preprocessing Overview

Data Cleaning

Handling missing values, removing outliers, fixing inconsistencies, and smoothing noisy data to improve quality.

Normalization & Scaling

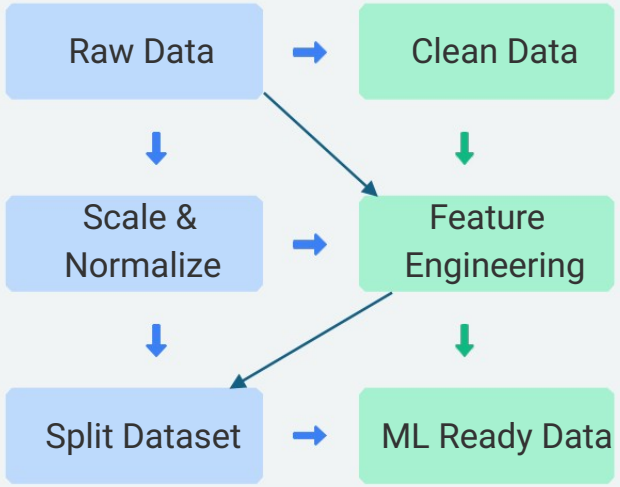
Adjusting values to a common scale (e.g., [0,1] for normalization, $\mu=0$, $\sigma=1$ for standardization) to prevent feature dominance.

Feature Engineering

Creating new features from existing ones to improve model performance, like frequency bands from time series or derived measurements.

Dataset Splitting

Dividing data into training, validation, and test sets (typically 70/15/15) to ensure unbiased model evaluation.



Normalization and Standardization

Min-Max Normalization

Scales features to a fixedrange [0,1]

$$x' = (x - min) / (max - min)$$

Preserves distribution shape, maintains relationships between values

Standardization (Z-score)

Transforms to mean=0, standard deviation=1

$$x' = (x - \mu) / \sigma$$

Usefulwhen features havevarying scales and/or outliers

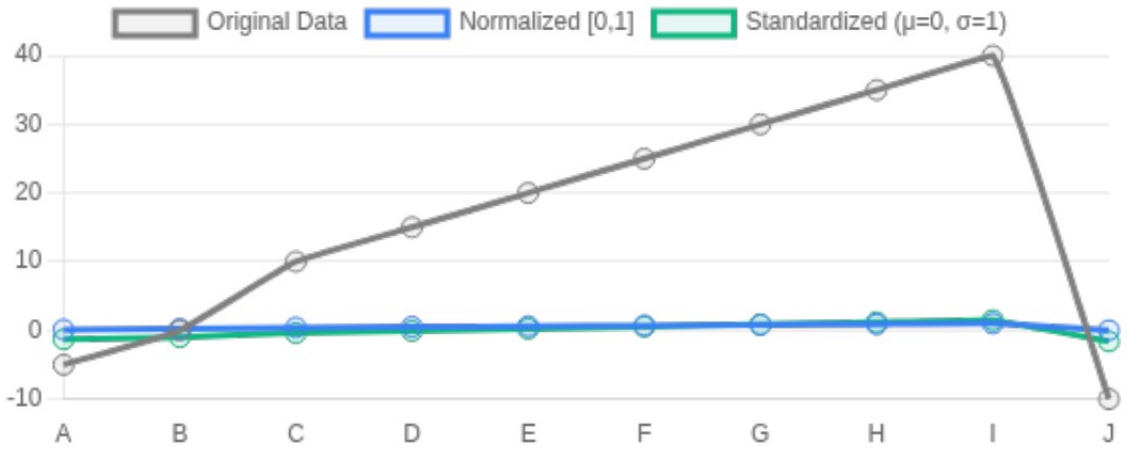
When to use Min-Max Scaling:

When you need values in predictable range (0-1)
For algorithms requiring bounded input
For image processing and neural networks

When to use Standardization:

For normally distributed features
For algorithms assuming zero-centered data
When outliers need to be handled well

Comparison: Original vs. Normalized vs. Standardized



Original Data	Normalized	Standardized
Range: [-10, 40]	Range: [0, 1]	Mean: 0, Std: 1

Python Code: Handling Missing Values

Essential techniques to identify and handle missingdata in machinelearning pipelines:

```
# Import necessary librariesimport
pandas as pd
import numpy as np
from sklearn.imputeimport SimpleImputer, KNNImputer

# Sample sensor data with missing values (NaN)data =
pd.DataFrame({
        'temperature': [85.0, 95.3, np.nan, 70.2, 99.8],
        'humidity': [45.2, np.nan, 32.1, 67.3, np.nan],
        'pressure': [1013, 1015, 1010, np.nan, 1009]
})

# 1. Identify missing values
print("Missing values per column:", data.isnull()sum())

# 2. Drop rows with missing valuesclean_data =
data.dropna()
print("\nAfter dropping rows:", len(clean_data), "rows remaining")

# 3. Fill missing values with mean (pandas)mean_filled =
data.fillna(data.mean())

# 4. Use sklearn SimpleImputer
imputer = SimpleImputer(strategy='mean') imputed_data =
pd.DataFrame(
        imputer.fit_transform(data),
        columns=data.columns
)
```

```
Missing values per column: temperature 1, humidity 2, pressure 1
After dropping rows: 1 rows remaining
Mean-imputed temperature: [85.0, 95.3, 87.58, 70.2, 99.8]
```

Pandas Approaches:

- df.isnull().sum(): Count missing values
- df.dropna(): Remove rows with NaN
- df.fillna(value): Replace NaN with value
- df.interpolate(): Interpolate missing values

Scikit-learn Imputers:

- SimpleImputer: mean, median, most_frequent
- KNNImputer: k-nearest neighbors
- IterativeImputer: Multivariate imputation
- Compatible with ML pipelines

Signal Processing Concepts – Frequency domain

Sampling & Nyquist Frequency

fs = 25,000 Hz

f_Nyquist = fs / 2 = 12,500 Hz

Max representable frequency without aliasing

FFT & Amplitude Spectrum

FFT converts time → frequency domain

Amplitude spectrum = |FFT(x)|

Shows energy at each frequency

Shaft Frequency

$$f_{shaft} = \frac{RPM}{60}$$

$$f_{shaft} = \frac{1238}{60} \approx 20.63 \text{ Hz}$$

Fundamental rotation frequency

Harmonics in Rotating Machinery

1X Harmonic

f = 20.63 Hz

Fundamental shaft frequency

Strong in **unbalance**

2X Harmonic

f = 41.27 Hz

Second harmonic

Strong in **misalignment**

Sub-harmonics

f < 20.63 Hz

Fractional multiples

Indicate **looseness**

Higher Multiples

3X, 4X, 5X ...

Integer multiples

Various fault indicators

Why Harmonics Matter for Fault Detection

Each mechanical fault excites the vibration signal at characteristic frequencies related to the shaft rotation speed:

Unbalance creates centrifugal force → dominant **1X** peak | **Misalignment** causes twice-per-revolution forcing → **1X + 2X** peaks

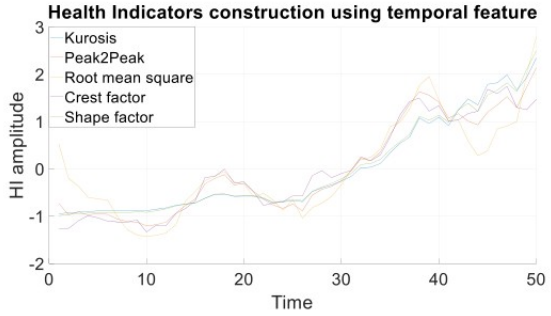
Looseness allows nonlinear motion → **sub-harmonics** and broad spectrum | **Normal** has low amplitude across all harmonics

Feature extraction

Signal processing techniques

Time domain

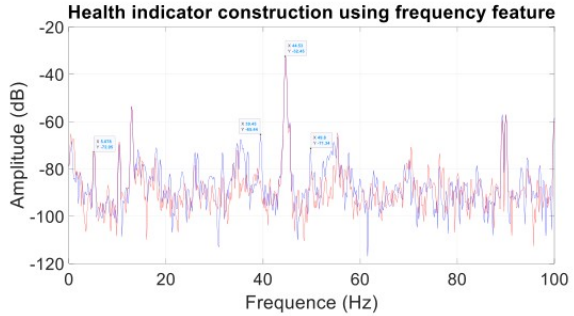
Root mean square, Mean, Crest factor, Peak-to-Peak, Kurtosis, ER, …



- ✓ Easy implementation and fast computing time
- ✓ Can reflect physical characteristics
- ✗ Difficult in non-stationary conditions
- ✗ Cannot identify the origin of the faults

Frequency domain

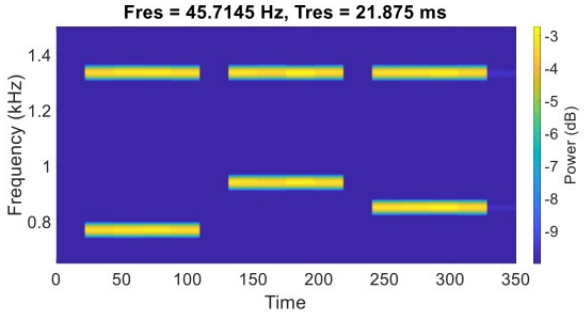
Fast Fourier Transform, Cepstrumtransform, Hilbert Transform, …



- ✓ Can detect and localize the origin of faults
- ✓ Fast computing time.
- ✗ Difficult in non-stationary conditions
- ✗ Require signal spectrum knowledge

Time-frequency dom

Fast Kurtogram, Hilbert Huang transform, Wavelet transform, …



- ✓ Appropriatefor non-stationary data
- ✓ Track fault frequency signatures over time
- ✗ Require high computational resolution
- ✗ Take a lot of computation times

Feature Engineering Overview

Why Convert Raw Signals to Features?

Raw signals have **25,000 dimensions per sample** — too many for efficient ML. Feature extraction reduces this to ~255 features while preserving fault-relevant information. This improves separability, reduces computation, and enables better generalization.

Feature Categories

Time-Domain Features

RMS — Root Mean Square: signal energy

Crest Factor — Peak/RMS: impulsiveness

Kurtosis — Distribution tail heaviness: shocks

Statistical measures of signal amplitude distribution

Frequency-Domain Features

1X Amplitude — Peak at shaft frequency

2X Amplitude — Peak at 2x shaft frequency

Band Energy Ratios — Energy distribution

Spectral measures capturing frequency patterns

Dimensionality reduction: 25,000 time points → 255 features = **98x compression** while preserving fault-discriminative information

Time-Domain Features

1. RMS — Root Mean Square

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}$$

Physical meaning: Signal energy

Sensitive to: Overall vibration level

Interpretation:

Higher RMS = more energy = likely fault

Increases with severity of most faults

2. Crest Factor

$$CF = \frac{max(|x_i|)}{RMS}$$

Physical meaning: Impulsiveness

Sensitive to: Sharp peaks and impacts

Interpretation:

High CF = spiky signal = impacts/shocks

Typical in looseness and bearing faults

3. Kurtosis

$$K = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i - \mu}{\sigma}\right)^4$$

Physical meaning: Tail heaviness

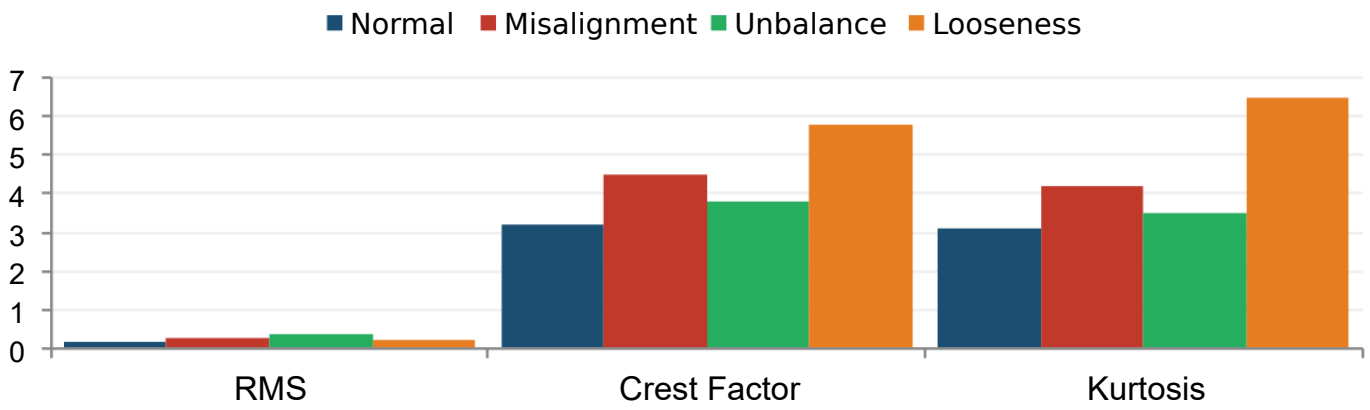
Sensitive to: Shocks and outliers

Interpretation:

Normal ≈ 3 (Gaussian), Fault >> 3

High kurtosis = heavy tails = shocks

Feature Comparison Across Conditions



Observations:

Unbalance has highest RMS (most energy)

Looseness has highest Crest Factor (most impulsive)

Looseness has highest Kurtosis (most shocks)

Normal has lowest values across all features

Frequency-Domain Features

FFT Magnitude Spectrum

Compute FFT of signal, take absolute value: fft_vals = np.abs(fft(x))

Total energy for normalization: E_total = sum(fft_vals^2)

1X Harmonic Peak

Max amplitude within ±2 Hz of f_shaft

$H_1 \approx 20.63$ Hz

Strong in **unbalance**

2X Harmonic Peak

Max amplitude within ±2 Hz of 2×f_shaft

$H_2 \approx 41.27$ Hz

Strong in **misalignment**

Band Energy Ratios

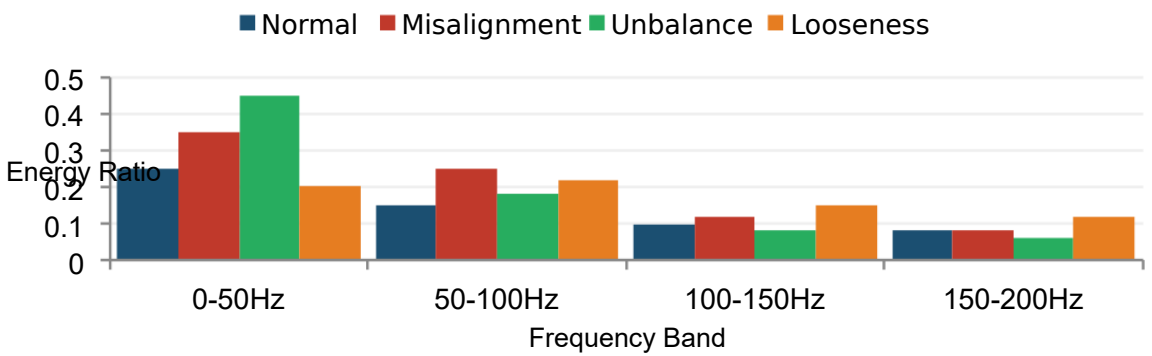
Divide spectrum into equal-width bands (e.g., 50 Hz)

For each band b: R_b = energy_in_band / (E_total + ε)

ε = 10^{-12} prevents division by zero

KEY EQUATION

$$R_b = \frac{\sum_{f \in band_b} (FFT(x)[f])^2}{E_{total} + \varepsilon} \text{ where } E_{total} = \sum_f (FFT(x)[f])^2 \text{ and } \varepsilon = 10^{-12}$$



Band Energy Insights:

Unbalance concentrates energy in low bands (0-50 Hz) due to strong 1X

Misalignment spreads energy across 0-100 Hz (1X + 2X)

Looseness has broader distribution across all bands

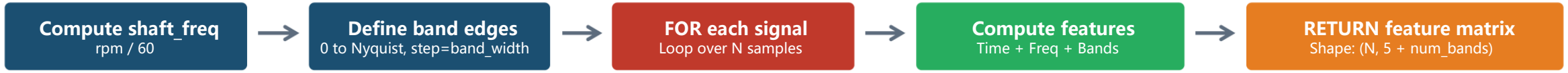
Normal has lowest energy ratios overall

Feature Extraction Function Walkthrough

Function Inputs

```
def Feature_extraction(signals, fs, rpm, band_width=50): # signals: (N, L), fs: 25000, rpm: 1238
```

Code Flow Diagram



Per-Signal Feature Computation

```
# === Time-Domain Features ===
rms = np.sqrt(np.mean(x** 2))
crest = np.max(np.abs(x)) / rms
kurt = np.mean(((x - mu)/std)** 4)
# === Frequency-Domain Features ===
fft_vals = np.abs(fft(x))
total_energy = np.sum(fft_vals** 2)
h1 = np.max(fft_vals[h1_mask]) # 1X
h2 = np.max(fft_vals[h2_mask]) # 2X
# === Band Energy Ratios ===
band_ratios = [sum(fft_vals[m]** 2) /
(total_energy + 1e-12)
for m in band_masks]
```

Output Feature Vector

Feature vector per sample:

[RMS, Crest, Kurtosis, 1X_Amp, 2X_Amp,

Band_Ratio_1, Band_Ratio_2, ..., Band_Ratio_N]

Total features: 5 + num_bands ≈ 255

Feature matrix shape:(2000, 255)

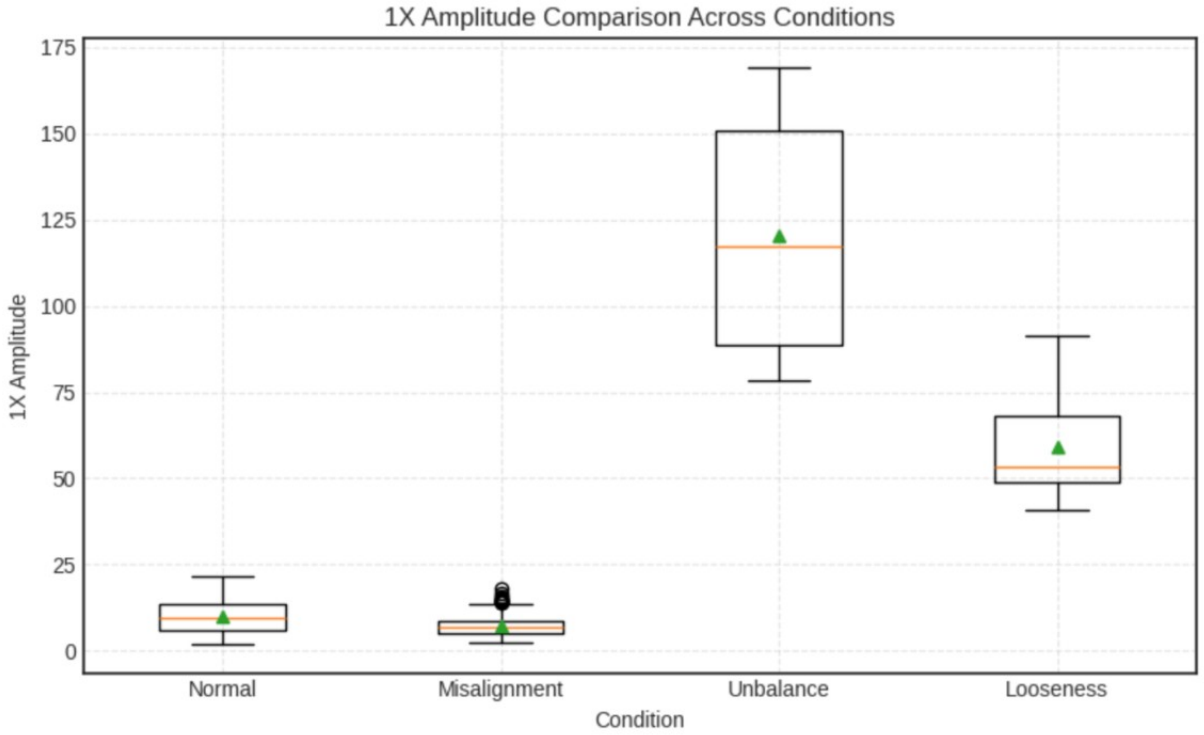
What the code does: Extracts 255 features per signal

Why needed: Reduces 25k points to discriminative features

Input: Raw signals (N×L), fs, rpm, band_width

Output: Feature matrix (N × 255)

Feature Visualization by Condition



How to Read Boxplots

Median — center line (typical value)

Box (IQR) — 25th to 75th percentile

Whiskers — spread of data

Outliers — points beyond whiskers

Good separation = boxes don't overlap much

Most Discriminative Features

1X Amp: Unbalance dominant

Conclusion: The extracted features show **clear separation** between conditions. Each fault type has a distinct signature across the feature set. This suggests that a classifier should be able to distinguish the four classes effectively.

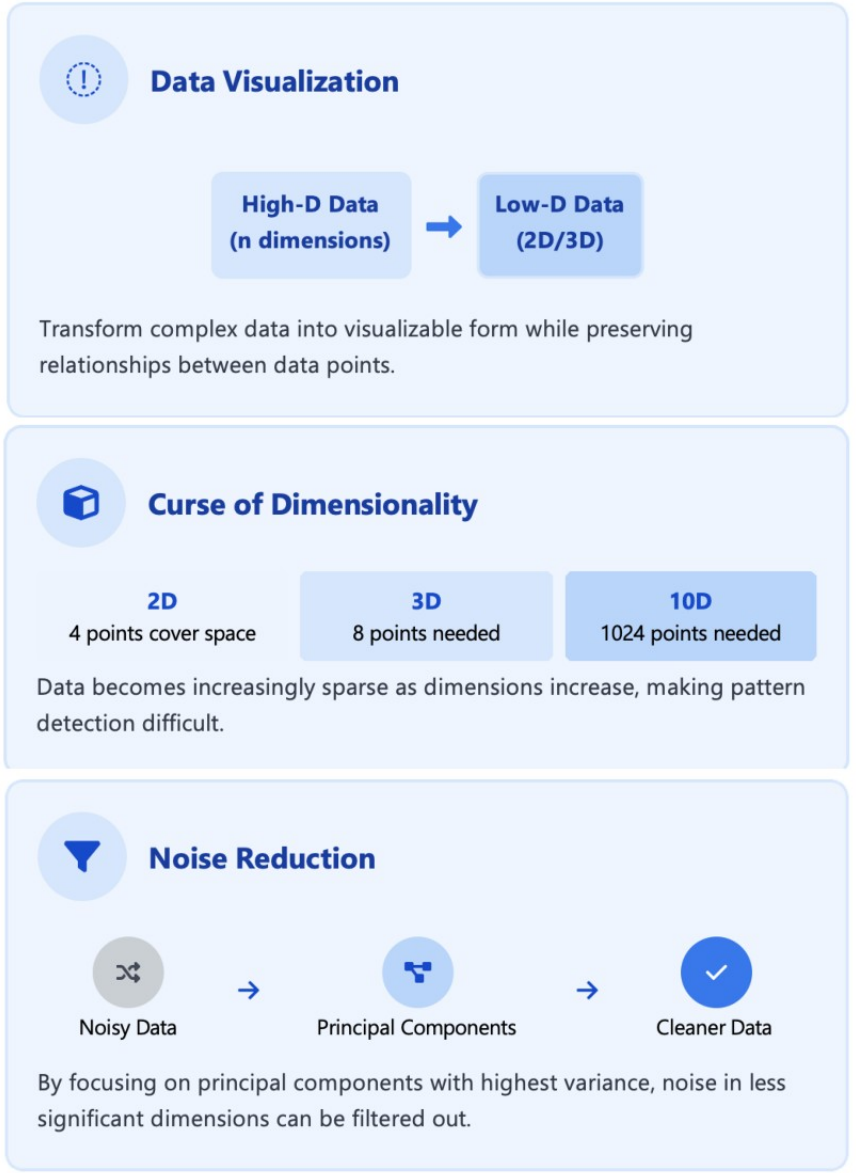
Dimensionality Reduction: Introduction

What is Dimensionality Reduction?

Dimensionality reduction transforms data from a high-dimensional space into a lower-dimensional space while preserving meaningful structure. It addresses several critical challenges in machine learning:

- **Data Visualization** - Reduce dimensions to 2D or 3D to visualize complex high-dimensional data
- **Curse of Dimensionality** - Combat sparsity issues and computational complexity in high-dimensional spaces
- **Noise Reduction** - Remove redundant features and noise to improve model performance
- **Feature Extraction** - Identify the most important underlying patterns in the data

> **Key Insight**
> Effective dimensionality reduction finds a balance between information preservation and complexity reduction, making previously intractable problems solvable.



Principal Component Analysis (PCA)

Mathematical Foundation

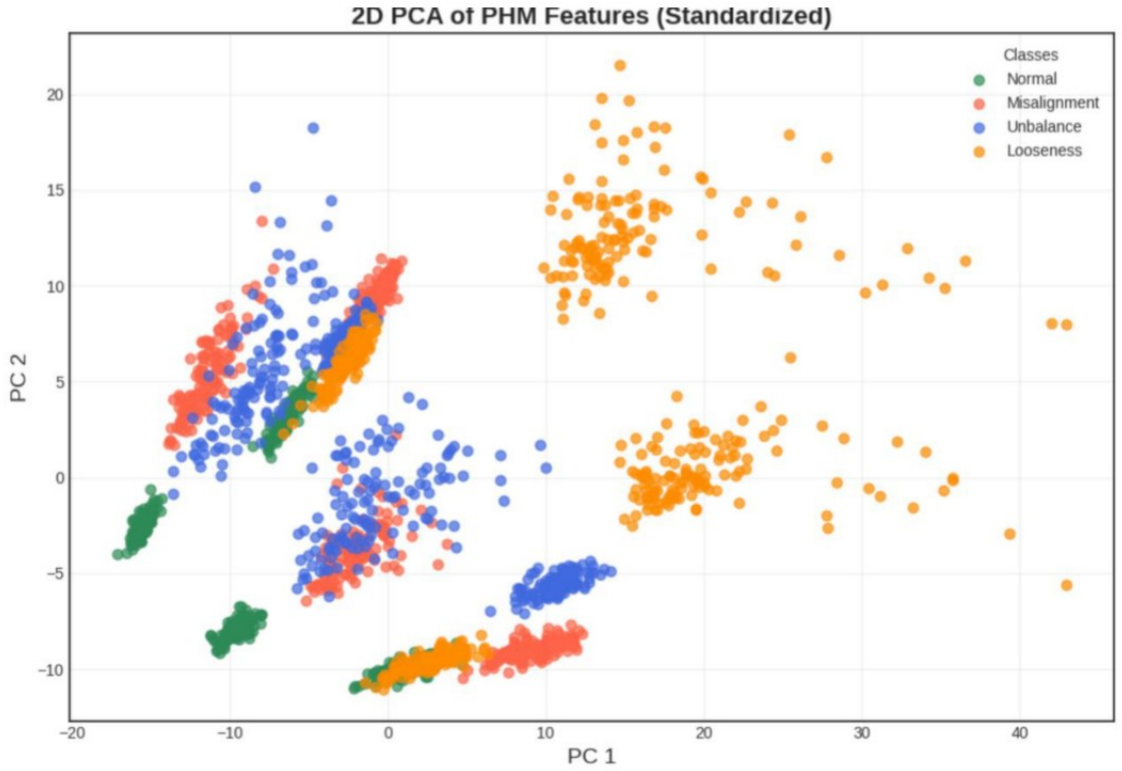
PCA is a linear dimensionality reduction technique that transforms data into a new coordinate system where the greatest variance lies on the first axis (first principal component), the second greatest variance on the second axis, and so on.

Key Steps:

1. Standardize the data (zero mean, unit variance)
2. Calculate the covariance matrix: $\Sigma = X^T X / (n-1)$
3. Calculate eigenvalues (λ) and eigenvectors (v) of Σ
4. Sort eigenvectors by decreasing eigenvalues
5. Select top k eigenvectors to form projection matrix W
6. Transform the original data: $X' = XW$

Key Insight

PCA is often used before applying machine learning algorithms to avoid the curse of dimensionality, speed up training, and reduce overfitting.



Explained Variance:

Each principal component captures a percentage of the total variance in the data. The explained variance ratio is: $\text{explained\_variance\_ratio}_i = \lambda_i / \Sigma\lambda_j$

Preparing Data for Classification

Step 1: Feature Stacking

```
X = np.vstack([X_normal, X_misalign, X_unbalance, X_looseness])# (2000, 255)
```

Step 2: Label Vector

```
y = [0]*N + [1]*N + [2]*N + [3]*N# 0=Normal, 1=Misalign, 2=Unbal, 3=Loose
```

Step 3: Train / Validation / Test Split

Stratified split ensures equal class proportions in each subset. random_state=42 for reproducibility.

Step 4: Feature Scaling

```
scaler = StandardScaler(); X_scaled = scaler.fit_transform(X)# zero mean, unit var
var
```

Step 5: One-Hot Encoding

```
y_cat = to_categorical(y)# [0,1,2,3] -> [[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]]
```

Data Preparation Summary

Step	Operation	Output Shape
Stack	np.vstack	(2000, 255)
Label	Concatenate	(2000,)
Split	Stratified	70/15/15%
Scale	StandardScaler	(N , 255)
Encode	to_categorical	(N , 4)
Final	Ready for model	X_scaled, y_cat

Why each step matters:

Stacking: Creates unified dataset

Labels: Enables supervised learning

Split: Prevents data leakage, enables unbiased evaluation

Scaling: Neural networks converge faster

One-hot: Required for softmax + cross-entropy

Machine Learning for Diagnostics

Objective in diagnostics
- Fault detection: *healthy vs faulty*
- Recognize **fault types** using extracted features

Commonly used classifiers
- **Logistic Regression**
  - Linear, probabilistic classifier
  - Fast, simple, and interpretable
- **k-Nearest Neighbors**
  - Instance-based learning
  - Classifies based on similarity between signals
- **Naive Bayes** Assumes feature independence
  - Very fast, effective with small datasets

Strengths
  - Easy to implement
  - Good baseline models

Non-linear and ensemble methods
- **Support Vector Machine (RBF kernel)**
  - Non-linear decision boundaries
  - High accuracy for complex signal features
- **Decision Tree**
  - Rule-based (if–then decisions)
  - Highly interpretable
- **Random Forest**
  - Ensemble of decision trees
  - Robust to noise and overfitting
  - Widely used in industrial diagnostics

Quick comparison
- **SVM** → High performance, sensitive to hyperparameters
- **Decision Tree** → Interpretable, prone to overfitting
- **Random Forest** → Best trade-off between accuracy and robustness

Autoencoder-Based Classifier Concept

Why Use an Autoencoder?

Combines **latent feature learning** (compression) + **reconstruction** (quality check) + **classification** (fault prediction) in a single model. The bottleneck learns a compact, nonlinear representation of the input features.

Model Specifications:

Input: 255 features (scaled)

Encoder: 255 → 128 → 64 → 32

Bottleneck: 32 (latent representation)

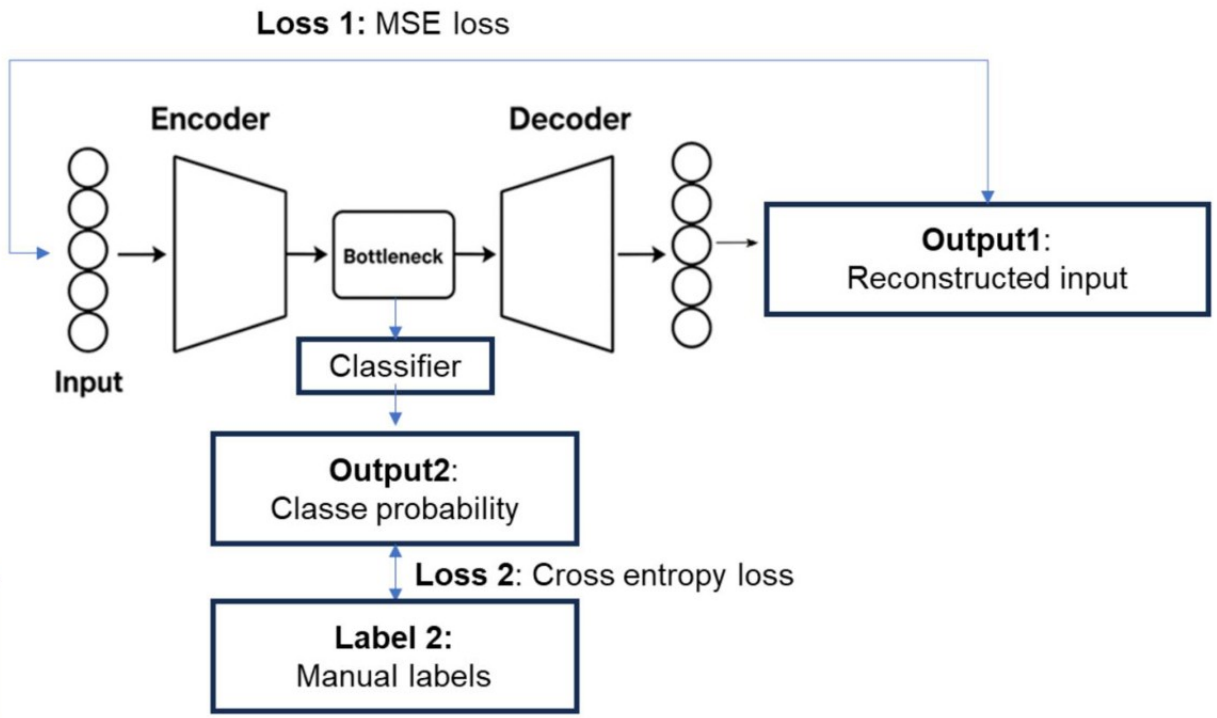
Decoder: 32 → 64 → 128 → 255

Classifier: 32 → 4 (softmax probabilities)

Total parameters: ~86,500

Activation: LeakyReLU (slope=0.1)

Regularization: Dropout (20%)



PCA vs. Autoencoder

PCA

Linear transformation | Finds orthogonal directions of max variance | Fast, interpretable | Limited to linear relationships

Autoencoder

Nonlinear transformation | Learns complex mappings via neural networks | Slower, more parameters | Captures nonlinear relationships

Model Training

Multi-Task Loss Function

Reconstruction Loss (MSE)

$L_{recon} = \frac{1}{N}\sum(x_i - \hat{x}_i)^2$

Weight: **0.5**

Classification Loss (Cross-Entropy)

$L_{cls} = -\sum y log(\hat{y})$

Weight: **1.0** (higher priority)

Total Loss

$L_{total} = 0.5 \times L_{recon} + 1.0 \times L_{cls}$

Adam optimizer, learning rate = 0.001

Training Callbacks

ModelCheckpoint

Saves best model when validation accuracy improves. File: best_model_diagnostics.h5

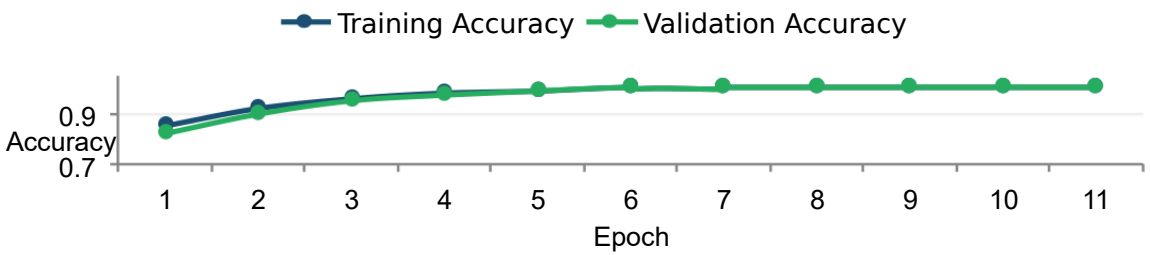
EarlyStopping

Stops if no improvement for 10 epochs. Restores best weights. Prevents overfitting.

Training Parameters

Parameter	Value
Epochs	50 (max)
Batch Size	32
O ptimizer	Adam (lr= 1e-3)
Early Stopping Patience	10 epochs

Training Curves



Observations:

Training converges

Validation accuracy

No overfitting (train ≈ validation)

Early stopping triggers at epoch 11

Evaluation Metrics

Classification Metrics

Accuracy
Correct / Total predictions
Overall correctness measure

Precision
True Positives / Predicted Positives
Reliability of positive predictions

Recall
True Positives / Actual Positives
Completeness of detection

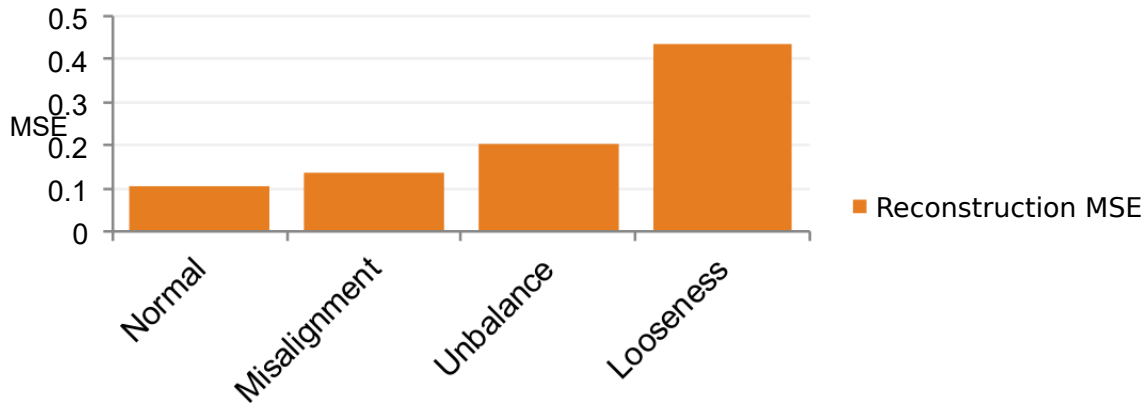
F1-Score
Harmonic mean of Precision & Recall
Balanced performance measure

Confusion Matrix

Rows = Actual, Columns = Predicted. **Diagonal = correct**, **Off-diagonal = errors**

	Pred N	Pred M	Pred U	Pred L
Act N	75	0	0	0
Act M	0	75	0	0
Act U	0	0	75	0
Act L	0	0	0	75

Reconstruction MSE by Class



Higher MSE = harder to reconstruct. **Looseness** has highest MSE, suggesting more complex patterns.

Metric	N ormal	Misalig nment	Unbalance	Looseness
Precision / Recall / F1	1.00 / 1.00 / 1.00	1.00 / 1.00 / 1.00	1.00 / 1.00 / 1.00	1.00 / 1.00 / 1.00
O verall Accuracy	100%			

Result Interpretation

TEST ACCURACY
100%

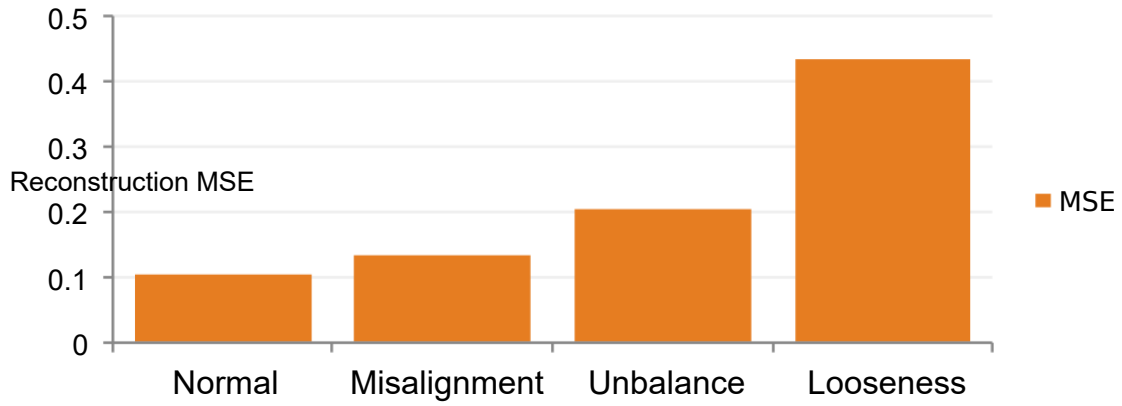
All **300 test samples** correctly classified

75/75 correct for each class: Normal, Misalignment, Unbalance, Looseness

Perfect precision, recall, and F1-score across all four fault types

Reconstruction Quality by Class

Condition	MSE	Interpretation
Normal	0.104	Easiest to reconstruct
Misalig nment	0.134	Low complexity
Unbalance	0.203	Moderate complexity
Looseness	0.433	Hig hest complexity



Important Caution

Perfect accuracy (100%) may indicate that the dataset is **highly separable** or that test conditions are very similar to training. For real-world deployment:

Validate on **broader unseen conditions** | Test with **different operating speeds** | Check **robustness to noise** | Evaluate **cross-machine generalization**

Positive insight: The reconstruction MSE pattern (Normal < Misalignment < Unbalance < Looseness) aligns with our physical understanding. More complex fault types are harder to reconstruct, confirming the autoencoder learns meaningful representations.

Outline

1. Introduction
2. Traditional maintenance strategies
3. Basic concepts of AI, and PHM
4. **From signals to decisions**
   - Diagnostics
   - **Prognostics**

Case Study Overview: C-MAPSS Turbofan Engines

C-MAPSS (Commercial Modular Aero-Propulsion System Simulation) is a**simulated turbofan simulated turbofan degradation dataset**from NASA, widely used as the benchmark for RUL benchmark for RUL prediction research.

Subset	Operating Conditions	Fault Modes	Difficulty	Engines (Train/Test)
FD001	1 (Sea Level)	1 (HPC Deg radation)	Easy	100 / 100
FD002	6 (Modes)	1 (HPC Deg radation)	Hard	260 / 259
FD003	1 (Sea Level)	2 (HPC + Fan)	Moderate	100 / 100
FD004	6 (Modes)	2 (HPC + Fan)	Hardest	248 / 249

Key Insight

Why generalization is difficult:

Each subset represents a different operational domain:

FD001 -- Single condition, single fault mode. Baseline benchmark.

FD002 -- Multiple conditions cause sensor distribution shifts.

FD003 -- Multiple fault modes create overlapping degradation patterns.

FD004 -- Combines all challenges: variable conditions + multiple faults.

Models trained on FD001 often fail on FD004 due to domain shift.

Predict the **Remaining Useful Life** (RUL) of turbofan engines under different degradation scenarios.

Data Dimensions per Engine:

- 21 sensor measurements
- 3 operational settings (altitude, Mach, throttle)
- Engine ID + cycle counter
- RUL label (target variable)

FD002 / FD004: Multiple Conditions

Engines operate under 6 different regimes. Same health state looks different under different settings. Requires condition-aware normalization (K-Means + per-condition scalers).

C-MAPSS Data Description

C olumn Group	C olumns	D escription
Identifier	unit_number	Eng ine id entifier (1 to 100)
Cycle	time_in_cycles	O perating cycle index fo r this eng ine
Setting s	setting _1, setting _2, setting _3	O perating conditio n variab les (altitud e, Mach, throttle)
Targ et	RUL	Remaining useful life (cycles until failure)
Sensors	senso r_1 to senso r_21	21 sensor measurements (temp , pressure, speed, flo w)
Total	26 columns	3 setting s + 21 sensors + 2 IDs/RUL
N ote	Some sensors are co nstant	Static senso rs can be removed during preprocessing
Fo rmat	Space-separated .txt	Each row = one cycle for one eng ine
Size	~ 100 eng ines per subset	Variable cycles p er eng ine (128 to 362)

Common Mistake: Do not treat each row as an independent sample. Each engine is a time series. All rows with the same unit_number belong to the same engine and must stay engine and must stay together during splitting.

Prognostics Challenges in the Tutorial

Heterogeneous Conditions

Operating conditions cause sensor distribution shifts. Same health state looks different under different settings. FD002/FD004 are especially affected.

Different Lifespans

Engines fail at different cycle counts (128 to 362). Model must generalize across short-lived and long-lived engines without memorizing individual patterns.

Nonlinear Degradation

Degradation is not linear. Some engines degrade gradually, others show sudden acceleration. Simple linear regression models often fail to capture these patterns.

Different RUL Progression Speeds

RUL decreases at different rates across engines. Some show rapid degradation near end-of-life; others degrade steadily from the start. Models must handle variable progression speeds without overfitting to specific patterns.

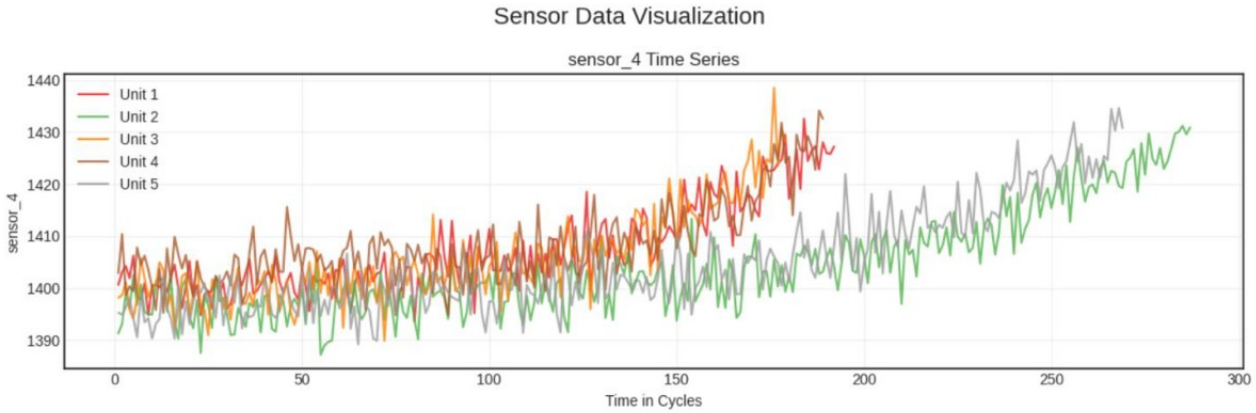
Multi-Domain Complexity

Each subset has distinct degradation curves and operating settings. Training on FD001 and testing on FD004 requires domain adaptation. Cross-subset evaluation is essential for claiming robust RUL prediction.

Key Takeaway: These challenges make RUL prediction fundamentally different from standard regression. The temporal structure, variable-length sequences, and domain shifts require specialized deep learning architectures and careful preprocessing.

Sensor Data Analysis — 21 Sensors, 3 Settings

Sensor C ateg ory	Sensors	Measurem ent
Temp erature	2, 3, 4, 7, 11, 12, 15	T2, T24, T30, T50, etc.
Pressure	6, 7, 10, 11, 12	P2, P15, P30, Ps30, EPR
Rotational S peed	8, 9, 13, 14, 18, 19	N f, N c, co rrected speeds
Fuel System	12, 16	Fuel flo w ratios
Eng ine Perfo rmance	15, 17, 20, 21	Bypass, bleed, efficiency
O perational Setting s	setting _1, 2, 3	A ltitude, Mach, throttle
To tal	21 sensors + 3 setting s	24 features + RUL



Key Insight: Not all sensors are informative. Some are static or noisy. Effective sensors reflect degradation and correlate with RUL. Joint learning across subsets is challenging due to distribution shift.

Source: NASA C-MAPSS Dataset Documentation

Training and testing data preparation

Sliding Windows in RUL Prognostics

Use **sliding windows** to convert long time-series engine data into shorter overlapping segments.
Each window: sensor behavior over a fixed time frame (e.g. 32 steps) and is labeled with the RUL at the window's endpoint.

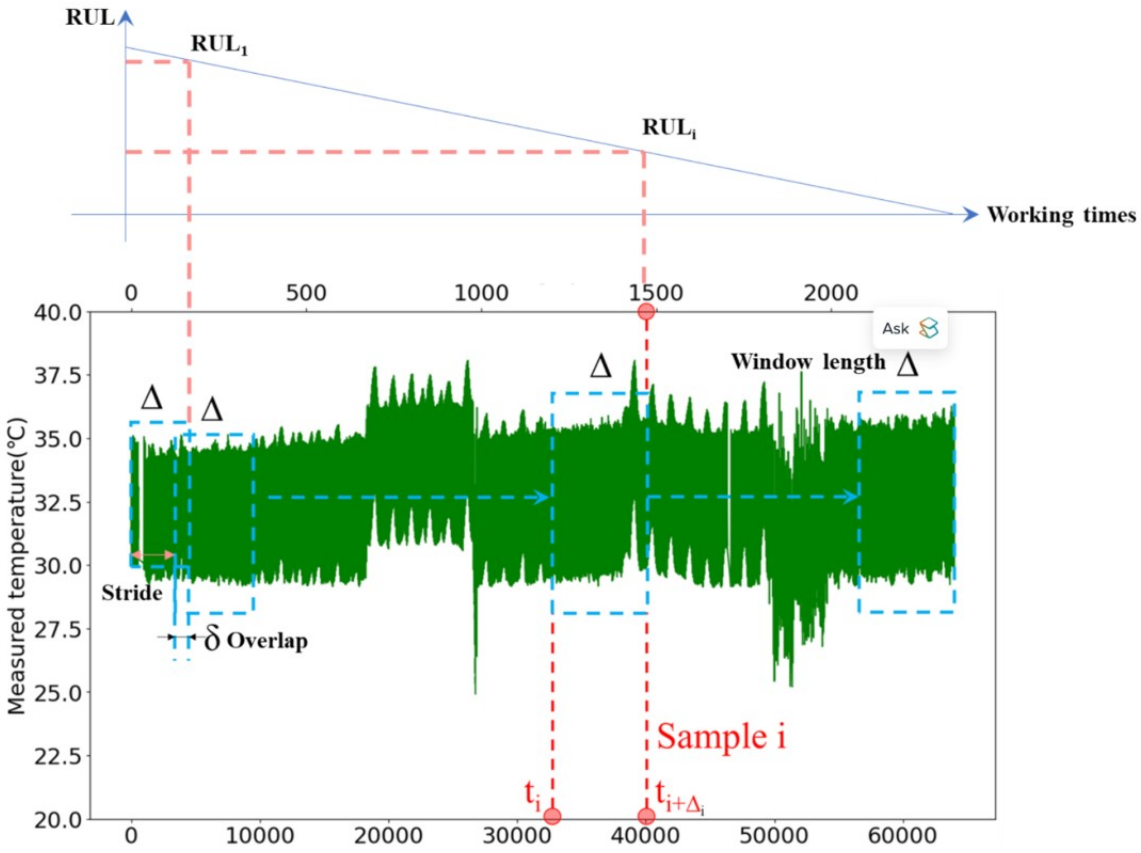
Why it's needed :
RUL prediction models (especially deep learning) require **fixed-size input samples**.
Sliding windows allow us to turn variable-length engine cycles into a large number of training examples that capture local degradation patterns.

How it works :

- For each engine, we slide a window over its time-series data with a given length and stride.
- Each window becomes one input sample (shape: `[window, features]`).
- The RUL label is taken from the last time step in the window.

Role in the Prognostics :

- Transform the raw dataset into a format ready for training neural networks
- Enable batch training, better generalization, and captures temporal features relevant to degradation and failure.



Split	Ratio	Purpose	Shuffle
Training	70%	Model learning	Yes
Validatio n	15%	H yperp arame ter tuning	N o
Test	15%	Final evaluatio n	N o

Batch Construction & Variable-Length Handling

Sliding Window Batching

Batch Shape: [batch_size, window_length, num_sensors]
Example: [64, 32, 21] = 64 windows x 32 cycles x 21 sensors

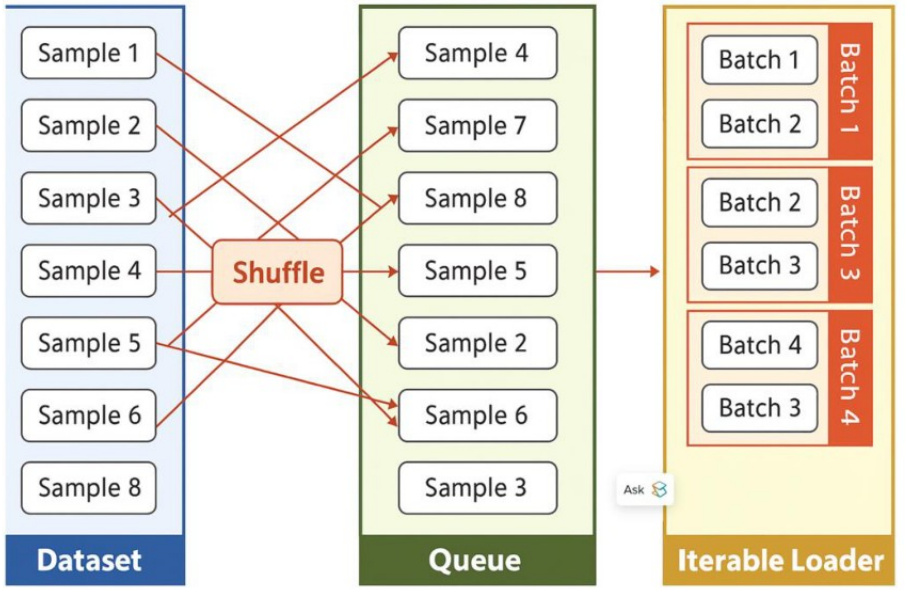
All windows have the same fixed size, so batching is straightforward with standard.

PyTorch DataLoader: No padding or masking needed.

Batching improves GPU utilization and stabilizes gradient estimates during training.

DataLoader: a pipeline that transforms raw data into structured mini-batches for model training.

- Start with a user-defined Dataset that specifies how to load each sample by index;
- Sampler generate the order of these indices (shuffled or sequential),
- BatchSampler group them into mini-batches
- Batches of indices are distributed to multiple parallel worker processes that call **getitem**() to fetch the actual data.
- Collate_fn stacks or pads the individual samples into properly aligned batch tensors.
- ensures that unstructured data (images, time series, text, etc.) is efficiently loaded, transformed, and batched for high-throughput model training.



Variable-Length Sequence Batching

For full sequences, engines have different lengths. We need:

1. Padding: Pad shorter sequences with zeros to match the longest in the batch.

2. Masking: Create a mask tensor (1 = valid, 0 = padded)# Masking : ignore padded positions in loss

Condition Normalization for Multi-Subset Learning

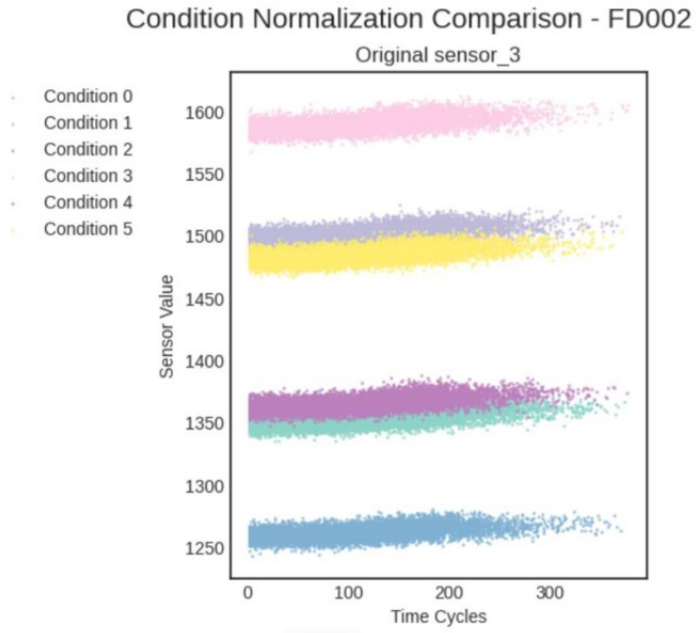
THE PROBLEM

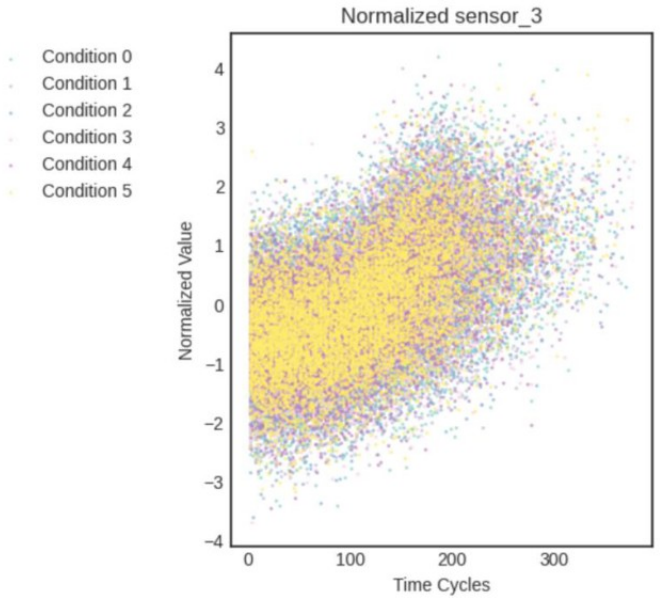
FD002 and FD004 have **multiple operating conditions** causing distribution shifts. Sensor readings at different altitudes/Mach numbers are not directly comparable. Merging subsets without normalization leads to poor model performance.

THE SOLUTION

KMeans clustering on operational settings identifies condition groups. **StandardScaler** normalization per condition group ensures consistent feature scales. Prevents data leakage by fitting scaler only on training data.

For FD001/FD003: Skip K-Means, use a single global StandardScaler. **For FD002/FD004:** Full pipeline with K-Means clustering + 6 separate StandardScalers. Each scaler learns the mean and std of learns the mean and std of sensors within its specific operating regime.





K-Means for Operating Condition Clustering

Goal: Cluster samples into operating condition groups based on the three

three setting variables.

Input features: setting_1, setting_2, setting_3

Why 6 clusters? FD002/FD004 have ~6 discrete operating regimes. K=6

approximates these regimes.

```
# K-Means clustering for operating settings
from sklearn.cluster import KMeans
settings_cols = [ 'setting_1' , 'setting_2' , 'setting_3' ]
kmeans = KMeans(n_clusters=6, random_state=42, n_init=10)
kmeans.fit(train_df[settings_cols])   # Fit on TRAINING only
train_df[ 'condition_id' ] = kmeans.predict(train_df[settings_cols])
val_df[ 'condition_id' ] = kmeans.predict(val_df[settings_cols])   # Apply to val
```

Pipeline Overview

settings -> K-Means -> condition_id

condition_id -> select scaler -> normalize

Fit K-Means on TRAIN only!

CORRECT: Fit on Train Only

```
# CORRECT: scaler knows only train
scaler.fit(train_data[sensors])  # train only
train_s = scaler.transform(train_data[sensors])
val_s = scaler.transform(val_data[sensors])
test_s = scaler.transform(test_data[sensors])
```

Applies to ALL preprocessing: StandardScaler, MinMaxScaler, K-Means, PCA, feature selection -- anything that computes statistics from data must be fit on training data only

Code Explanation

What it does: Groups samples into 6 operating regimes using the 3 setting variables.

Why it is needed: Different operating conditions produce different sensor baselines. Normalizing within each condition removes this confounding factor.

Input: setting_1, setting_2, setting_3 (3 columns)

Output: condition_id column (0 to 5) for each sample

Prognostics Model Family

Model	H ow It W orks	Streng ths	W eaknesses
MLP	Flatten features, feed through dense layers	Simple, fast, good baseline	Loses temporal structure
CN N	1D convolutions over tim e axis	Captures local temporal patterns	Limited receptive field
LSTM / GRU	Recurrent g ates preserve long -term memory	Proven for deg radation sequences	Sequential processing is slow
A utoencoder	Encoder -> latent -> decoder + RUL head	Learns useful representations	Multi-task training complexity
LSTM + A ttention	LSTM + weig hted importance over timesteps	Focuses on key cycles + interpretable	Slightly more param eters
Transform er	Self-attention over all tim esteps	Long -rang e dependencies, parallel	N eeds more data, compute-heavy

This tutorial focuses on: LSTM (baseline), Autoencoder (representation learning), **and BiLSTM + Attention (focus + interpretability.)** These three models provide a progression from provide a progression from simple to advanced with increasing capability.

Baseline LSTM for RUL Prediction

Architecture: Input -> LSTM -> FC -> RUL

```
class LSTM_RUL(nn.Module):
def __init__(self, input_dim, hidden_dim):
super().__init__()
self.lstm = nn.LSTM(input_dim, hidden_dim,
batch_first=True, num_layers=2)
self.fc = nn.Linear(hidden_dim, 1)
def forward(self, x):
lstm_out, (h_n, _) = self.lstm(x)# [B, T, H]
rul = self.fc(h_n[-1])# last hidden
return rul.squeeze(-1)
```

How LSTM Works for RUL

Input: [batch, time, sensors] e.g., [64, 32, 21]

Processing: LSTM reads sequence step by step, updating hidden state at each cycle.

Memory: Forget gate discards old info; input gate adds new info; output gate produces hidden state.

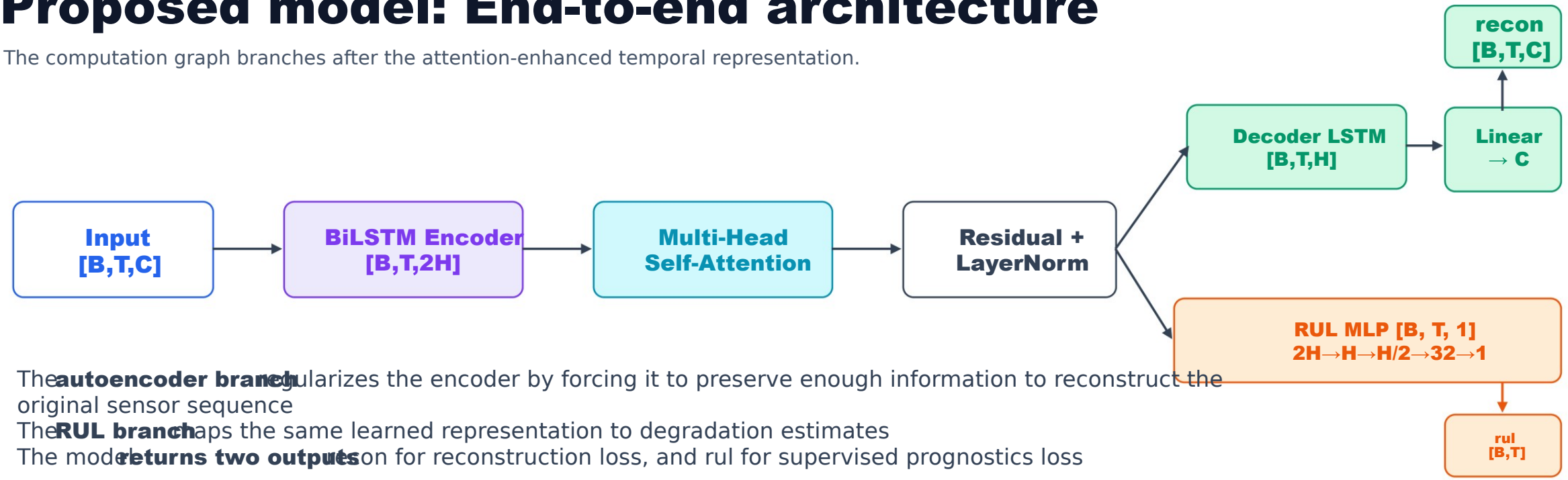
Output: Final hidden state h_n[-1] summarizes full sequence -> FC layer -> single RUL value.

Why it works: Hidden state accumulates degradation evidence over time. By the end, it encodes enough info to predict remaining life.

Key Point: The LSTM uses only the final hidden state for prediction. All intermediate hidden states are discarded. This is a limitation that attention mechanisms address by using ALL hidden states with learned importance weights.

Proposed model: End-to-end architecture

The computation graph branches after the attention-enhanced temporal representation.



The **autoencoder branch** regularizes the encoder by forcing it to preserve enough information to reconstruct the original sensor sequence
The **RUL branch** maps the same learned representation to degradation estimates
The model **returns two outputs**: recon for reconstruction loss, and rul for supervised prognostics loss

C = input_size / sensor channels; T = sequence length; H = hidden_size; 2H appears because the encoder is bidirectional

```
encoder_outputs, _ = self.lstm_encoder(x)
attn_output, _ = self.attention(encoder_outputs, encoder_outputs, encoder_outputs)
encoder_outputs = self.layer_norm(encoder_outputs + attn_output)
```

Tensor shapes through the forward pass

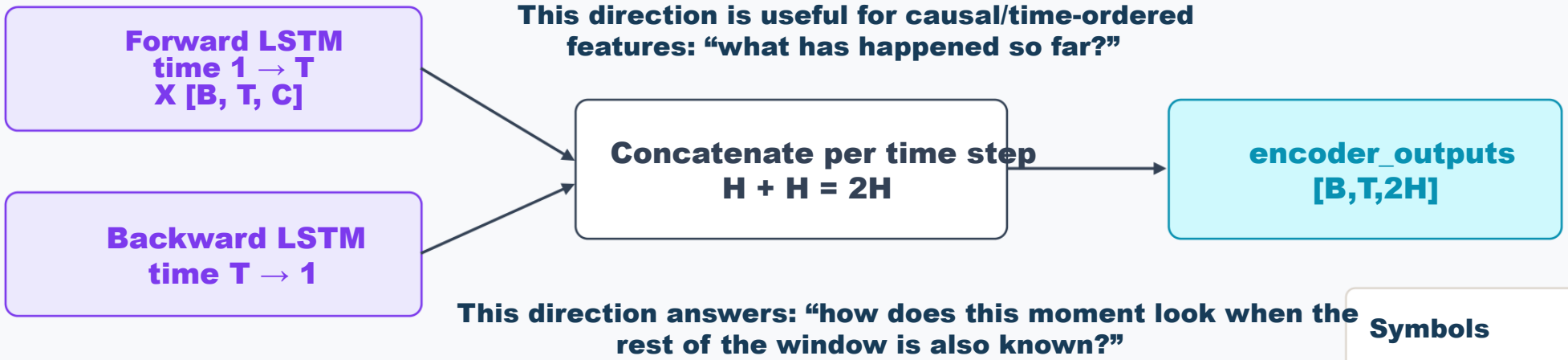
Shape consistency is the backbone of this model.

Stage	Operation	Shape
Input	x	[B, T, C]
Encoder	bidirectional LSTM	[B, T, 2H]
Attention	MHA(Q=K=V=encoder_outputs)	[B, T, 2H]
Residual norm	LayerNorm(encoder + attention)	[B, T, 2H]
Decoder	LSTM + Linear	[B, T, C]
RUL head	MLP + squeeze(-1)	[B, T]

Example: B=64, T=50, C=14, H=128 ⇒ encoder output is [64, 50, 256].

Encoder: stacked bidirectional LSTM

The encoder extracts temporal features from sensor windows.



Symbols

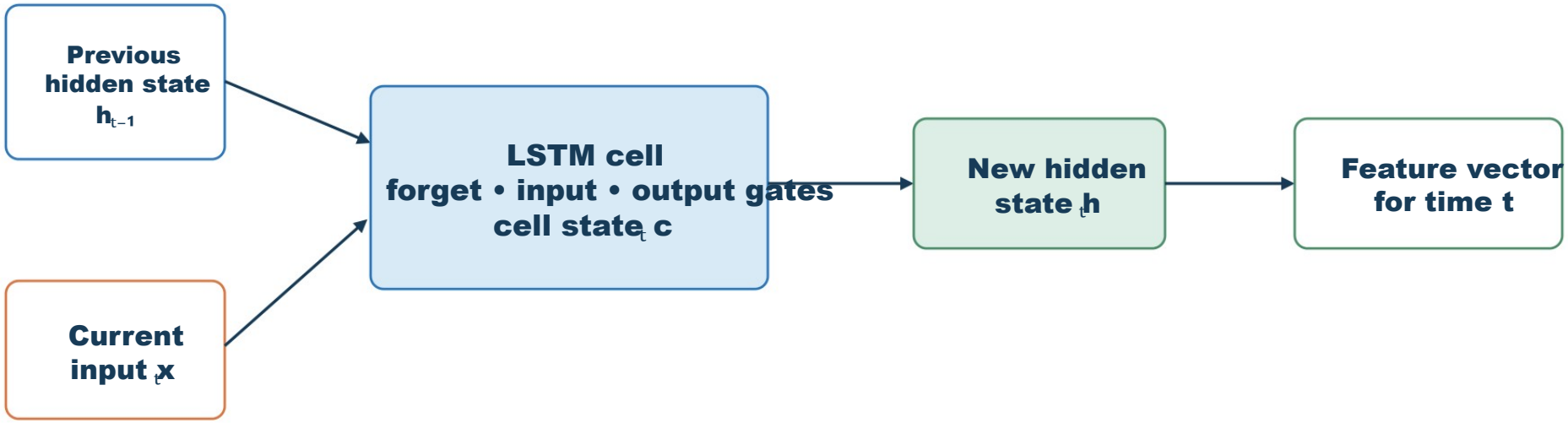
B = batch size
T = sequence length (window length)
C = input_size, number of sensor channels
H = hidden_size per direction
2H = forward H + backward H

num_layers=3 stacks multiple recurrent layers, increasing temporal abstraction.
dropout=0.3 is applied between LSTM layers during training.
bidirectional=True allows each time step to use both earlier and later context inside the window.
The model uses encoder_outputs, not only the final hidden state.

```
self.lstm_encoder = nn.LSTM(input_size, hidden_size,
    num_layers=3, batch_first=True, dropout=0.3, bidirectional=True)
```

What an LSTM Cell Learns at Each Time Step

An LSTM is designed to preserve useful historical information while filtering irrelevant noise.



Gate intuition

Forget gate: what past information should be discarded?
Input gate: what new information should be stored?
Output gate: what should be exposed as h_t?

Why this matters for RUL

Sensor degradation is temporal. A current reading is meaningful only when interpreted against previous vibration, temperature, pressure, or other sensor trends.

In your model

Every time step in the sensor window is passed through these recurrent computations before attention is applied.

Stacked BiLSTM: num_layers = 3

Your encoder uses multiple recurrent layers to build increasingly abstract temporal features.

Layer 1

Learns short-term transitions: sudden spikes, local changes, short operating cycles.

Layer 2

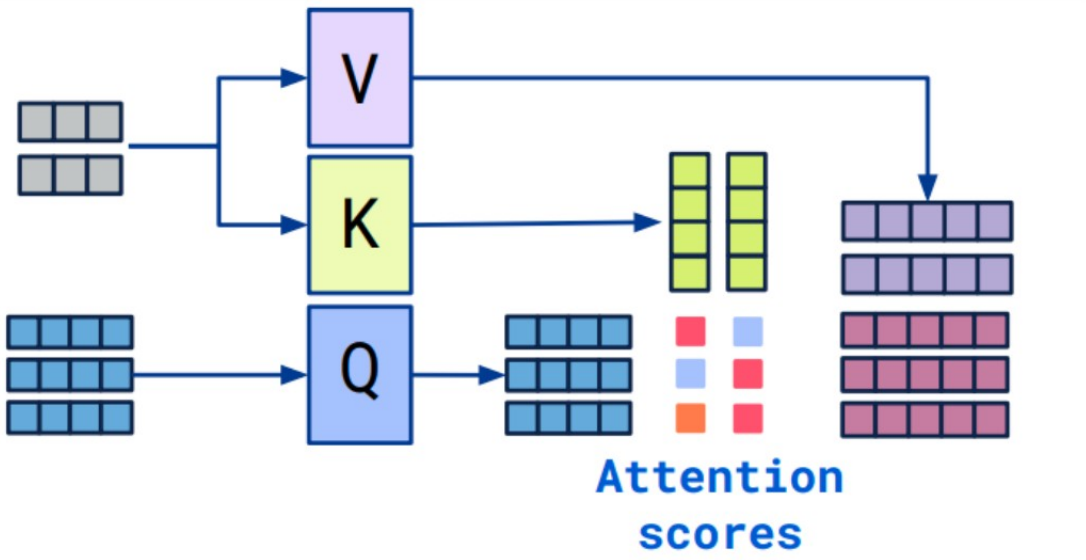
Combines lower-level states into broader temporal motifs: rising vibration after temperature increase.

Layer 3

Builds compact degradation context useful for attention, reconstruction, and RUL estimation.

Dropout is applied between LSTM layers during training to reduce overfitting.

Self-Attention — Query, Key, and Value



Self-attention computes attention scores between **all pairs of positions** in a sequence sequence simultaneously.

Three Learned Projections:

- **Query (Q)** — What am I looking for?
- **Key (K)** — What do I contain?
- **Value (V)** — What information do I provide?

Attention Formula:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Multi-Head Attention: Multiple attention operations in parallel, attending to different representation subspaces.

Intuition: Each position in the sequence creates a query and compares it against all keys. High similarity = high attention weight. The output is a weighted sum of values, where weights are the attention scores. This allows each position to "attend to" all other positions in a single operation.

Source: The Illustrated Transformer — Jay Alammar, Vaswani et al. (2017)

Where Q, K, V come from in our model

Before attention, the input passes through the BiLSTM encoder

- x.shape = [batch, time, input_size]
  #x.shape = [32, 50, 14]: 32 samples per batch; 50 time steps per sample; 14 sensor channels/features per time step
- encoder_outputs.shape = [batch, time, hidden_size * 2] (due to BiLSTM)
  #hidden_size = 128: each time step now has a 256-dimensional representation
- self.attention(encoder_outputs, encoder_outputs, encoder_outputs) #Q = K =V = encoded LSTM sequence

Query, or Q, represents: What is this time step looking for?

- For each time step, the model creates a query vector. That query asks: Which other time steps are relevant to me?
  Which previous time steps show similar degradation?
  Which earlier readings indicate the start of failure?
  Which time steps contain abnormal vibration?
  Which time steps are important for estimating remaining life?

Key, or K, represents: What information does each time step contain, in a searchable form?

- Each time step also gets a key vector. The key is used for matching against queries
- If the query from time step 45 strongly matches the key of time step 30, then the model treats time step 30 as *important for understanding time step 45*

Value, or V, represents: What actual information should be passed forward if this time step is important?

- Key is used to decide relevance while value contains the information that gets combined into the attention output
- If time step 45 attends strongly to time steps 30, 38, and 44, then the output for time step 45 becomes a weighted combination of vectors from those time steps
- new representation at time 45 = 0.01 × V1 + 0.02 × V2 + … + 0.22 × V30 + 0.30 × V38 + 0.28 × V44

Signification of Q, K, V come from in our model

Suppose your sequence has 5 time steps instead of 50, just for illustration

```
Time step | Sensor behavior
----------|---------------------------
t1        | normal vibration, normal temperature
t2        | slight vibration increase
t3        | temperature begins rising
t4        | vibration spike
t5        | high vibration + high temperature
```

After the BiLSTM, each time step becomes an encoded vector:

$E1, E2, E3, E4, E5$

Your attention layer receives:

$Q = [E1, E2, E3, E4, E5]$
$K = [E1, E2, E3, E4, E5]$
$V = [E1, E2, E3, E4, E5]$

Now suppose the model is computing the attention output for t_5.

The query is Q_5.

It compares Q_5 with all keys:

$Q_5 \cdot K_1$
$Q_5 \cdot K_2$
$Q_5 \cdot K_3$
$Q_5 \cdot K_4$
$Q_5 \cdot K_5$

Maybe the model produces these similarity scores:

```
t1: 0.2
t2: 0.4
t3: 0.7
t4: 0.9
t5: 1.0
```

After softmax, these become attention weights:

```
t1: 0.08
t2: 0.10
t3: 0.19
t4: 0.30
t5: 0.33
```

Then the new representation for t_5 becomes:

```
Attention output for t5 =
    0.08 × V1
  + 0.10 × V2
  + 0.19 × V3
  + 0.30 × V4
  + 0.33 × V5
```

This means the model decided: For RUL prediction, time step 5 may need informatic from earlier warning signals at time steps 3 and 4

Multi-head attention: multiple temporal relations in parallel

The 2H feature vector is split across 8 attention heads.

Each head learns attention in a different subspace — local trends, repeated cycles, anomaly points, long-range degradation, or operating-regime shifts.

PyTorch constraint: (hidden_size * 2) must be divisible by num_heads.
With H=128 and 8 heads, embed_dim=256 and each head uses 32 dimensions.

```
self.attention = nn.MultiheadAttention(embed_dim=hidden_size*2, num_heads=8, dropout=dropout, batch_first=True)
```

How Q, K, V are actually computed inside model

nn.MultiheadAttention learns three separate linear projections

Q = encoder_outputs × W_Q
K = encoder_outputs × W_K
V = encoder_outputs × W_V

Model learns W_Q, W_K, and W_V during training

In our model:

Q.shape = [32, 50, 256]
K.shape = [32, 50, 256]
V.shape = [32, 50, 256]

Multi-head attention splits the 256 features into 8 heads:

head_dim = 256 / 8 = 32

Each head computes its own attention matrix: [32, 50, 50]

- For each sample in the batch, each time step compares with all 50 time steps

After all 8 heads are computed, their outputs are concatenated back

attn_output.shape = [32, 50, 256]
encoder_outputs = self.layer_norm(encoder_outputs + attn_output)

How this helps your RUL model

rul_features = encoder_outputs

For example, for predicting RUL at t50, the model may attend to:
t5 : early healthy baseline
t20 : first degradation signal
t35 : vibration increase
t45 : temperature rise
t50 : current degraded state

This gives the RUL head more context than using only the LSTM state at t50

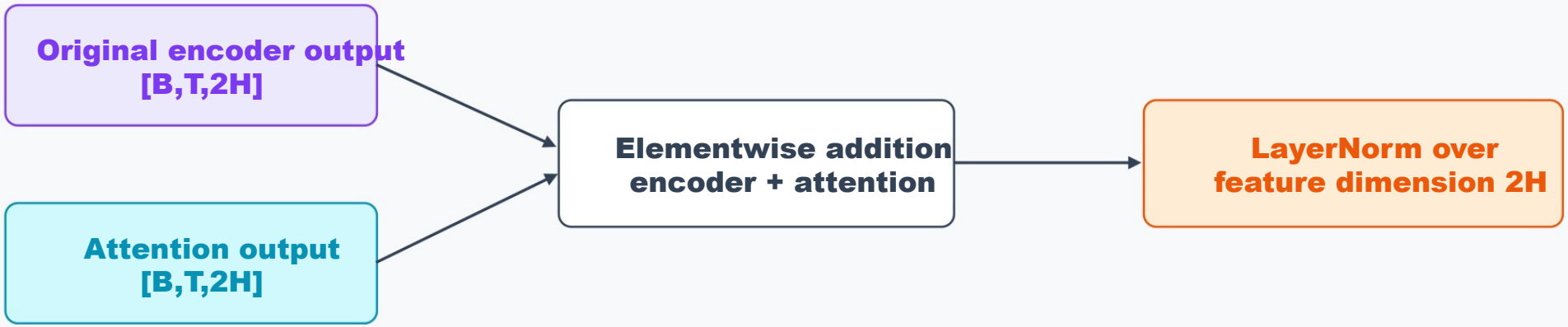
Decoder branch: reconstruct the input sensor sequence

The autoencoder objective encourages the shared representation to retain signal structure.


```
decoder_outputs, _ = self.decoder_lstm(encoder_outputs)
recon = self.decoder_output(decoder_outputs)
```

Typical reconstruction loss: MSE(recon, x).
Because the decoder receives the full attended sequence, this is a contextual sequence reconstruction module, not a strict bottleneck-only autoencoder.
The branch can improve robustness when RUL labels are noisy by forcing representations to encode sensor dynamics.

Residual connection + LayerNorm




```
encoder_outputs = self.layer_norm(encoder_outputs + attn_output)
```

Residual path: Attention does not overwrite the recurrent representation; it adds contextual corrections.
LayerNorm keeps the scale: of hidden features more consistent, improving optimization stability.
The output remains the shared representation used by both decoder and RUL head.

RUL prediction head: Dense regression at each time step

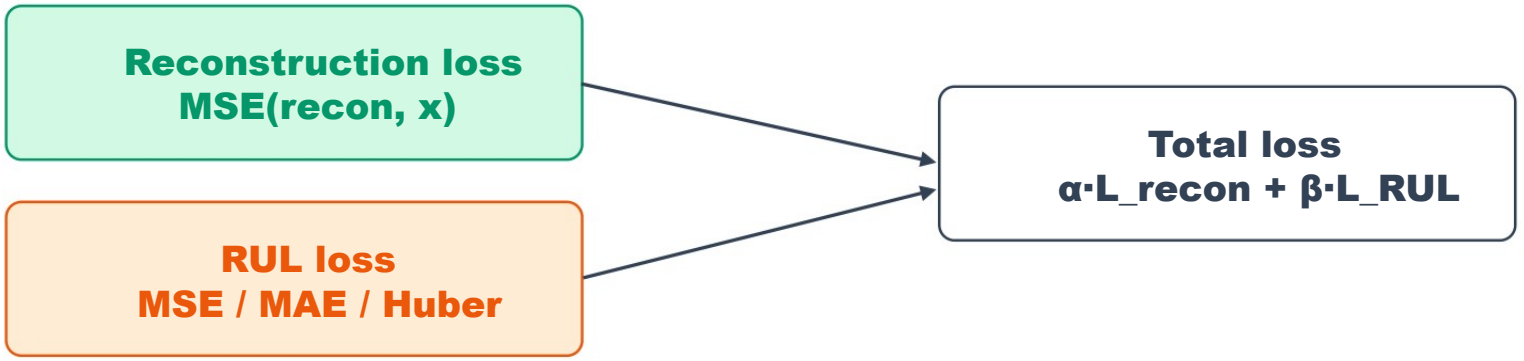
A multilayer perceptron maps attention-enhanced features to scalar life estimates.


```
rul = self.rul_layers(encoder_outputs).squeeze(-1)
# before squeeze: [B,T,1]
# after squeeze:  [B,T]
```

The final Linear(32,1) performs scalar regression for each time step.
If labels are sequence-level [B], use rul[:, -1] or temporal pooling before computing loss.
If labels are time-step-level [B,T], use the full output sequence directly.

Training objective: combine reconstruction and prognostics

The total loss should reflect both tasks and label granularity.




```
loss_recon = F.mse_loss(recon, x)
loss_rul   = criterion(rul, target_rul)      # if target_rul: [B,T]
# or criterion(rul[:, -1], target_rul)       # if target_rul: [B]
loss = alpha * loss_recon + beta * loss_rul
```

Balance α and β carefully: too much reconstruction can dominate supervised RUL learning

Evaluation Procedure & Metrics

Evaluation Procedure

1. Load best model checkpoint
2. Set **model.eval()**
3. Wrap in **torch.no_grad()**
4. Predict on test_loader
5. Compare predicted vs true RUL
6. Compute metrics

Metric	Form ula	Interpretation	W hen to Use
MSE	$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$	Penalizes large errors strong ly. Lower is better.	Standard comparison
RMSE	$RMSE = \sqrt{\frac{\sum_{i=1}^{N}\|\|y(i) - \hat{y}(i)\|\|^2}{N}}$	Error in same unit as RU L (cycles). Most interpretable.	Primary reported m etric
MAE	$MAE = \frac{1}{n}\sum_{i=1}^{n}\|y_i - \hat{y}_i\|$	A verag e absolute error. Robust to outliers.	N oisy data, outliers
Tolerance A cc	predictions within [-13, + 10] cycles	Operational relevance. % of acceptable predictions.	PH M applications

References

Documentation & Tutorials

- Kamal Medjaher. (*IGENI-EC0931) Prognostics and Health Management Methods –Tools – Applications*. [Course materials]. Univ. Toulouse, UTTOP.
- Khanh T. P. Nguyen. *Introduction to Artificial Intelligence*. [Course materials]. Univ. Toulouse, UTTOP.
- Khanh T. P. Nguyen, Kamal Medjaher, Do Tran. *A review of artificial intelligence methods for engineering prognostics and health management with implementation guidelines* , Artificial Intelligence Review 56 (4), 3659-3709, 2023.
- Vaswani et al., Attention Is All You Need, NeurIPS 2017
- Andrew Ng. *AI For Everyone* (Coursera)
- Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning*, MIT Press, 2016
- scikit-learn - Official Documentation (scikit-learn.org)
- Python Data Science Handbook by Jake VanderPlas