Eastern Switzerland
University of Applied Sciences

Master's Thesis

Automated Modeling for Predictive Maintenance

Comparison and Implementation of AutoML, Agent-Based and LLM-Based approaches for RUL Prediction

Joshoua Bigler

Prof. Dr. Daniel Lenz — *Advisor*
Prof. Dr. Thomas Koller — *Co-Advisor*

January 21, 2026

Abstract

Predictive maintenance (PdM) relies on accurate estimation of the remaining useful life (RUL) of industrial components to reduce unplanned downtime and maintenance costs. While data-driven approaches have shown strong performance, their practical deployment is often hindered by the need for substantial machine learning expertise. Automated machine learning (AutoML) addresses this challenge by reducing manual modeling effort, but remains constrained by fixed search spaces and limited adaptability. Recent agentic artificial intelligence (AI) approaches based on large language model (LLM) promise greater flexibility through automated pipeline generation and reasoning; however, their effectiveness for RUL prediction is not yet well understood.

This thesis presents a systematic comparison of three automated modeling strategies for RUL prediction: a standalone AutoML baseline, a fully agentic approach based on LLM, and a hybrid approach combining agentic guidance with AutoML. All approaches are evaluated on the NASA C-MAPSS turbofan engine dataset and the FEMTO bearing dataset using run-to-failure sensor data. The comparison considers predictive performance, robustness, failure behavior, and practical deployment aspects.

The results show that classical AutoML provides reliable and stable baseline performance, but is constrained by predefined search spaces and limited ability to incorporate domain knowledge, which restricts its adaptability. Standalone agentic approaches partially alleviate these limitations, but exhibit increased failure rates, reduced robustness, and lower predictive accuracy. In contrast, the hybrid agentic AutoML approach consistently combines the strengths of both paradigms by enabling the incorporation of domain knowledge, the construction of dataset-specific search spaces, and the generation of new regression models, while achieving competitive predictive performance (RMSE) and maintaining robustness. These findings indicate that agentic AI is most effective when used in combination with structured AutoML workflows.

Taken together, the results suggest that the agentic hybrid approach can lower the barrier to deploying predictive maintenance systems, enabling small and medium-sized businesses (SMEs) to benefit from data-driven RUL prediction with less specialised in-house expertise.

Acknowledgements

I would like to thank Prof. Dr. Daniel Lenz for his supervision and constructive input throughout the course of this thesis. His feedback helped shape the direction and clarity of the work. I also thank Prof. Dr. Thomas Koller for serving as co-advisor and evaluator of this work .

Declaration of Authorship

I, Joshoua Bigler, declare that this thesis titled, 'Automated modeling for Predictive Maintenance Comparison and Implementation of AutoML, Agent-Based and LLM-Based approaches for RUL Prediction' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master's degree at the OST - Eastern Switzerland University of Applied Sciences.
- Where any part of this report has previously been submitted for a degree or any other qualification at this university of applied science or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this work is entirely my own work.
- I have acknowledged all main sources of help.
- Where the report is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Joshoua Bigler	January 21, 2026
	Date

Contents

Chapter 1

Introduction

This thesis focuses on predictive maintenance (PdM) for small and medium-sized enterprises (SMEs) in the mechanical engineering industry. Among established maintenance approaches, PdM offers the most cost-efficient strategy, as it maximises machine uptime through early fault detection (Bigler 2025b, p. 7–8). Despite strong industrial interest, early industry surveys indicate that the adoption of advanced predictive maintenance remains limited, largely due to organizational and analytical barriers (PwC 2017). Key challenges include the availability of suitable data, the required analytical expertise, and the complexity of developing reliable predictive models.

To address these barriers, this thesis proposes an automated PdM framework for remaining useful life (RUL) prediction of industrial machinery components. The framework is evaluated using run-to-failure sensor data from the C-MAPSS turbofan engine (Arias Chao et al. 2021) and FEMTO bearing (Nectoux et al. 2012) benchmark datasets. While earlier work addressed an Internet of Things (IoT) based Software as a Service (SaaS) platform (Bigler 2025b; Bigler 2025a), the present thesis focuses specifically on automated modeling approaches for PdM.

This work investigates three complementary modeling paradigms: a classical automated machine learning (AutoML) baseline, a standalone agentic artificial intelligence (AI) approach, and a hybrid agentic AutoML system. The agentic approaches leverage large language models (LLMs) as a core reasoning component to incorporate user-provided domain knowledge and to generate dataset-specific machine learning pipelines and regression models, rather than relying on fixed, predefined configurations.

The objective of this thesis is to examine the capabilities and limitations of such automated modeling frameworks and to systematically compare classical AutoML, standalone agentic approaches, and hybrid agentic AutoML pipelines for RUL prediction. The evaluation focuses on predictive performance measured by root mean square error (RMSE) on benchmark datasets, as well as robustness, generalizability across datasets and the required development effort.

Chapter 2

Theoretical Background and Related Work

This chapter provides the theoretical foundations and related work underlying this thesis. It introduces core concepts of maintenance, AutoML, and agentic AI to establish the context for the methodology presented in the following chapters.

2.1 Maintenance

2.1.1 Maintenance Strategies

Maintenance is a critical factor in the machine industry, affecting operational costs and system reliability (Bigler 2025b, p. 7). Unplanned machine downtime can lead to significant financial losses and potentially cause damage to a company's reputation.

- **Reactive maintenance:** Repairs are performed only after failure, causing high costs and long downtimes.
- **Preventive maintenance:** Maintenance is carried out at fixed intervals, preventing some failures but often leading to unnecessary work.
- **Predictive maintenance:** Uses real-time data to predict failures and schedule maintenance efficiently, reducing unplanned downtime and unnecessary servicing.

PdM offers the most cost-effective maintenance strategy by effectively balancing preventive actions and repair expenses.

2.1.2 Predictive Maintenance

According to Bigler (2025b, pp. 9–10), PdM estimates the time to failure using real-time data to eliminate unexpected downtime, improve overall reliability and reduce operating costs. Approaches are generally classified into:

- **Physical model-based:** Uses domain knowledge to build mathematical degradation models; offers interpretability but exhibits limited scalability.

- **Data-driven:** Relies on statistical and machine learning techniques; well-suited for complex systems but harder to interpret.
- **Hybrid:** Combines physical insights with data-driven methods for improved accuracy and generalization.

Developing robust systems that detect, diagnose, and adapt to diverse operating conditions remains challenging. Variability between machines, limited failure data, and high data quality requirements make modeling difficult. Temporal and spatial dependencies further increase complexity, making reliable and interpretable PdM systems challenging to design in practice.

Remaining Useful Life

The remaining useful life represents the time interval between the current time and the expected failure of a system. Within predictive maintenance, RUL estimation constitutes a central prognostic task, providing quantitative information to support maintenance planning and decision-making.

Remaining useful life estimation can be addressed using model-based, data-driven, or hybrid approaches. Model-based methods rely on physical or mathematical degradation models and require detailed domain knowledge, which limits their applicability to complex systems. Data-driven approaches, in contrast, infer degradation patterns directly from sensor data using machine learning techniques, reducing the need for explicit system models (Bigler 2025b). Formally, for a system observed at time t, the remaining useful life can be defined as:

$$\mathrm{RUL}(t) = t_f - t \tag{2.1}$$

where t_f denotes the failure time according to a predefined failure criterion. In data-driven settings, RUL estimation is typically formulated as a supervised sequence-to-target learning problem (Zöller et al. 2023, p. 2). The goal is to learn a mapping $h : X \to Y$, where X represents the space of multivariate time series $\vec{x}_i$ describing the system's condition, and $Y \subset \mathbb{R}$ the corresponding RUL values y_i. The training dataset is given by $D_{\text{train}} = \{(\vec{x}_i, y_i)\}_{i=1}^{N}$, and the test dataset by $D_{\text{test}} = \{(\vec{x}'_i, y'_i)\}_{i=1}^{M}$. Each instance $\vec{x}_i$ contains sensor measurements from the start of operation until a defined failure criterion. Model performance is evaluated using a loss function L on D_{test}. For the final evaluation, the loss is computed on the test dataset D_{test}. In this work, the root mean square error is used as the evaluation metric, as it is the standard performance measure used in RUL benchmark datasets:

$$L_{\text{RMSE}}(h, D_{\text{test}}) = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(h(\vec{x}'_i) - y'_i)^2} \tag{2.2}$$

with $h(\vec{x}'_i)$ being the predicted RUL for the instance $\vec{x}'_i$.

2.2 Time Series Data

As described in Darban et al. (2025, p. 3), a time series represents a collection of observations arranged in chronological order and recorded at successive time intervals. Time series data can be categorized as *univariate*, involving a single variable, or *multivariate*, involving multiple variables. A univariate time series (UTS) with t time steps is defined as:

$$X = (x_1, x_2, \ldots, x_t) \tag{2.3}$$

A multivariate time series (MTS) consists of d dimensions, each corresponding to a distinct sensor value, observed over time. At each time point i, the system state is represented by a vector X_i containing all d sensor values:

$$X = (X_1, X_2, \ldots, X_t) = ((x_1^1, x_1^2, \ldots, x_1^d), (x_2^1, x_2^2, \ldots, x_2^d), \ldots, (x_t^1, x_t^2, \ldots, x_t^d)) \tag{2.4}$$

Time series data exhibits temporal dependencies and, in the multivariate case, spatial dependencies. These characteristics distinguish time series data from conventional tabular data and require models that explicitly account for temporal and spatial dependencies.

2.3 Automated Machine Learning for RUL Prediction

Developing data-driven RUL prediction models typically requires substantial expertise in machine learning, feature engineering, and data preprocessing. However, as noted by Zöller et al. (2023, p. 1), such expertise is often scarce in SMEs, which limits the practical deployment of data-driven PdM solutions. AutoML for RUL prediction aims to reduce this entry barrier by enabling domain experts to automatically construct and optimize RUL models without extensive machine learning knowledge.

2.3.1 Motivation for AutoML

In practice, datasets differ greatly between applications. Sometimes only a few sensors are available, while other systems record dozens of measurements at varying sampling rates, noise levels, and operating conditions. No single model performs optimally across all such cases; for instance, deep neural networks may excel on large multivariate datasets, whereas random forests often perform better when only limited features are available. AutoML addresses this challenge by automatically selecting and tuning suitable models for the given data, reducing manual effort and improving efficiency in model development for diverse use cases.

2.3.2 Hyperparameter Optimization

A core component of AutoML is the automation of model selection and hyperparameter tuning. Given a learning algorithm $\mathcal{A}$ with hyperparameter space $\Lambda = \Lambda_1 \times \cdots \times \Lambda_N$, the hyperparameter

optimization (HPO) objective can be formulated as (Hutter, Kotthoff, and Vanschoren 2019, p. 84):

$$\lambda^* \in \arg\min_{\lambda \in \Lambda} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\Big(\mathcal{A}_\lambda, D_{\text{train}}^{(i)}, D_{\text{valid}}^{(i)}\Big) \tag{2.5}$$

where $\mathcal{L}$ denotes the validation loss over k folds.

2.3.3 Combined Algorithm Selection and Hyperparameter Optimization

Equation 2.5 formalizes hyperparameter optimization for a single, fixed algorithm. The combined algorithm selection and hyperparameter optimization (CASH) problem extends this formulation by jointly determining both the learning algorithm and its hyperparameters (Hutter, Kotthoff, and Vanschoren 2019, p. 84):

$$A^*, \lambda^* \in \arg \min_{A^{(j)} \in \mathcal{A},\, \lambda \in \Lambda^{(j)}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\Big(A_\lambda^{(j)}, D_{\text{train}}^{(i)}, D_{\text{valid}}^{(i)}\Big) \tag{2.6}$$

This unified formulation forms the mathematical foundation of most AutoML systems. By simultaneously optimizing algorithm choice and hyperparameters, it enables automatic discovery of both the most suitable learning method and its configuration for a given dataset.

2.3.4 Bayesian Optimization

Hyperparameter optimization in AutoML is typically framed as a *black-box* optimization problem: given a configuration $\lambda \in \Lambda$, the objective function $\mathcal{L}(A_\lambda, D_{\text{train}}, D_{\text{valid}})$ can be evaluated, but its analytical form and gradients are unknown. Evaluating this function is often computationally expensive, as it involves training and validating a model.

Bayesian optimization (BO) provides a principled strategy for optimizing such expensive black-box functions (Hutter, Kotthoff, and Vanschoren 2019, pp. 9–12). It constructs a probabilistic *surrogate model* $\mathcal{M}_\mathcal{L}$ that approximates the relationship between configurations and their observed losses, including uncertainty estimates. Based on this model, an *acquisition function* selects promising configurations by balancing exploration of uncertain regions and exploitation of regions expected to perform well.

In practice, Bayesian optimization is commonly realized through sequential model-based optimization (SMBO), which explicitly implements this procedure and is discussed in the following section.

2.3.5 Sequential Model-Based Optimization

A widely used strategy for solving the CASH problem (Equation 2.6) is Bayesian optimization, and in particular sequential model-based optimization (Thornton et al. 2013). SMBO is a stochastic optimization framework that is well suited to black-box objective functions and can naturally handle

categorical, continuous, and conditional hyperparameters.

As illustrated in algorithm 1, SMBO iteratively constructs and refines a probabilistic *surrogate model* $\mathcal{M}_{\mathcal{L}}$ of the objective function $\mathcal{L}(\lambda)$. This surrogate provides a computationally cheap approximation of the true validation loss and is used to guide the search towards promising algorithm and hyperparameter configurations.

Algorithm 1: SMBO, adapted from Thornton et al. (2013, p. 2)

Input: Time budget T
Output: Best configuration $\lambda^\star$
Initialize model $\mathcal{M}_L$;
$\mathcal{H} \leftarrow \emptyset$;
while *time budget for optimization has not been exhausted* **do**
  $\lambda \leftarrow$ candidate configuration from $\mathcal{M}_L$;
  Compute $c = \mathcal{L}(A_\lambda, D^{(i)}_{\text{train}}, D^{(i)}_{\text{valid}})$;
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\lambda, c)\}$;
  Update $\mathcal{M}_L$ given $\mathcal{H}$;
end
return λ from $\mathcal{H}$ with minimal c;

At each optimization step, SMBO maintains a surrogate model $\mathcal{M}_{\mathcal{L}}$ that approximates the relationship between a configuration λ and its validation loss $\mathcal{L}(\lambda)$. Here, a configuration $\lambda \in \Lambda$ jointly specifies the learning algorithm and its associated (conditional) hyperparameters. The surrogate serves as a cheap-to-evaluate proxy for the expensive true objective. Based on this model, SMBO selects a new candidate configuration λ by maximizing an *acquisition function* $a_{\mathcal{M}_{\mathcal{L}}}(\lambda)$, which balances *exploration* (testing uncertain regions) and *exploitation* (refining well-performing ones):

$$\lambda = \arg\max_{\lambda \in \Lambda} a_{\mathcal{M}_{\mathcal{L}}}(\lambda) \tag{2.7}$$

After selecting λ, the true objective

$$c = \mathcal{L}(A_\lambda, D_{\text{train}}, D_{\text{valid}}) \tag{2.8}$$

is evaluated on the actual learning algorithm. The resulting observation (λ, c) is added to the history of evaluated configurations, and the surrogate model is retrained on this expanded dataset $\mathcal{H}$ to form the updated model $\mathcal{M}_{\mathcal{L}}$. This iterative process continues until a termination condition, such as a time or evaluation budget, is reached.

2.3.6 Multi-fidelity Optimization

Increasing dataset sizes and increasingly complex models pose a major hurdle for hyperparameter optimization, as they make black-box performance evaluations computationally expensive. Training a single configuration on large datasets can take hours or even days. To mitigate this, *multi-fidelity*

optimization methods exploit *low-fidelity approximations* of the objective function, such as training on smaller data subsets, fewer epochs, or reduced feature sets to obtain cheap but informative performance estimates. These methods trade off evaluation accuracy against runtime, but in practice the resulting speedups usually outweigh the approximation error (Hutter, Kotthoff, and Vanschoren 2019, p. 14).

Successive halving (SH) (Jamieson and Talwalkar 2015) formalizes this principle as an adaptive resource allocation strategy. It starts by evaluating many configurations on a small initial budget and iteratively allocates more resources to the best-performing fraction while discarding poorly performing ones. At each stage, only the top $1/\eta$ of configurations are promoted, and their budgets are multiplied by η. This process efficiently concentrates computation on promising configurations and achieves strong theoretical guarantees compared to uniform budget allocation.

Hyperband (HB) (Lisha Li et al. 2018) extends SH through a *hedging strategy* that automatically balances exploration and exploitation. Instead of committing to a single allocation schedule, it runs multiple SH instances (*brackets*) with different trade-offs between the number of configurations and their per-configuration budgets.

As stated in Hutter, Kotthoff, and Vanschoren (2019), to overcome Hyperband's random sampling limitation, Bayesian optimization and Hyperband (BOHB) combines Bayesian optimization with Hyperband's multi-fidelity scheduling. It replaces random sampling with a model-based proposal strategy, achieving both sample and resource efficiency and demonstrating strong performance across diverse optimization tasks.

2.3.7 State of the Art AutoML for RUL Predictions

Zöller et al. (2023) demonstrated the viability of applying AutoML to RUL prediction. Their framework, Automated Machine Learning for Remaining Useful Life Predictions (AUTORUL), achieved at least competitive results across a wide range of datasets and outperformed its predecessor proposed by T. Tornede et al. (2021). No other state-of-the-art model was able to generalize similarly well across all benchmarks. Since most existing approaches are tailored to specific datasets, this finding highlights that a universal, one-size-fits-all model remains difficult to construct. Instead, automatically generating simple models for each dataset can be a more effective and efficient strategy.

2.3.8 Limitations of Current AutoML Approaches

Despite their progress, current AutoML systems still face several limitations that constrain their practical applicability. First, as noted by Zöller et al. (2023, p. 6), the fixed and predefined search space prevents such systems from becoming truly universal tools, since it will always be possible to construct a dataset that lies outside their representational capacity. Moreover, to unleash their full potential, AutoML frameworks still require careful configuration and parameterization, often demanding expert knowledge to operate effectively (A. Tornede et al. 2024, p. 11). Finally, as emphasized by A. Tornede et al. (2024, p. 12), most AutoML tools offer only limited user interaction,

which restricts their ability to incorporate human intuition or domain expertise during the search process.

2.4 Agentic AI

As noted by Krishnan (2025, pp. 1–7), an AI agent is a system capable of autonomously performing tasks on behalf of a user or another system by planning its own workflow and using available tools. Such agents extend beyond natural language processing, incorporating decision-making, problem-solving, environmental interaction, and autonomous execution. Recent advances in LLM have given rise to a new category often referred to as LLM-based agents, or agentic AI, which integrate modules for memory, planning, tool use, and environmental interaction.

2.4.1 AI Agent Architecture and Components

According to Krishnan (2025, pp. 11–14), modern AI agent architectures integrate multiple components that enable autonomous perception, reasoning, and action. Although implementations differ across domains, several core modules consistently form the foundation of effective agent design.

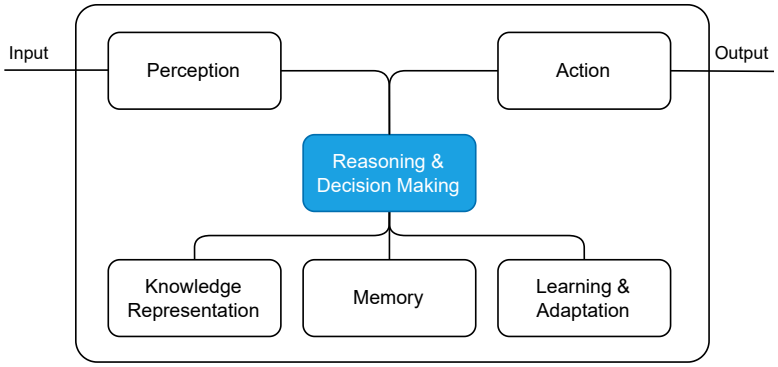


Figure 2.1: AI Agent Architecture, own illustration based on Krishnan (2025, p. 11)

Perception Mechanisms
Serves as the interface to the environment, transforming raw inputs, such as language, sensor data, or images, into structured representations for further processing.

Knowledge Representation Systems
Knowledge representation systems provide the structures for storing, organizing, and retrieving information within an agent. They must balance expressiveness (capturing diverse knowledge types), computational efficiency (enabling fast access and manipulation), and learnability (supporting continuous updates). Modern architectures employ *hybrid approaches* that combine symbolic structures, such as ontologies, knowledge graphs, and logical rules, with distributed representations like vector embeddings. These systems typically distinguish between declarative (facts about the world), procedural (how to perform specific tasks), episodic (records of specific experiences), and meta-knowledge (information about the agent's own capabilities and limitations).

Reasoning and Decision-Making Modules

Reasoning and decision-making modules enable agents to interpret information, evaluate alternatives, and select appropriate actions. As noted by Scharei, Heidecker, and Bieshaar (2020, p. 2), reasoning involves solving problems through logical deduction and thus relies on available knowledge. These modules employ various forms of inference, including *deductive* (deriving logical consequences from premises), *inductive* (generalizing from observations), *abductive* (inferring plausible explanations), and *analogical reasoning* (applying insights from similar past situations). The integration of large language models has significantly enhanced reasoning capabilities. Nevertheless, their reasoning remains imperfect, and advanced agent architectures often augment LLMs with specialized reasoning modules to improve robustness. Decision making components then transform reasoning outputs into concrete actions, commonly using utility-based approaches that balance expected outcomes with the agent's objectives.

Action components

Action components turn an agent's decisions into concrete behaviors, such as producing language output or invoking tools. They often rely on hierarchical action structures, where high-level goals are broken down into smaller steps that can be executed effectively.

Learning and Adaptation Mechanisms

Learning and adaptation mechanisms allow agents to improve through experience. They can refine perception with supervised learning, optimize actions through reinforcement learning, discover structure using unsupervised or self-supervised methods, and adapt more quickly via meta-learning. Feedback from other agents or humans further guides this continuous improvement.

Memory management

Memory mechanisms help agents retain information across interactions, preserving context and enabling agents to learn from experience. They often distinguish working, episodic, semantic, and procedural memory, each serving a different role in storing and organizing experience.

2.4.2 LLMs for AutoML

As stated by A. Tornede et al. (2024, pp. 12–14), it is expected that LLMs will disrupt the field of AutoML from various angles. In particular, three main opportunities are highlighted:

1. **Opportunity I: Improving Human-Machine Interaction**
   (a) Interface: LLMs can serve as user-friendly interfaces to AutoML, enabling users to provide domain knowledge.
   (b) Interpretability: LLMs can generate clear textual explanations of optimization histories, improving the transparency of AutoML and providing user-friendly reports.

2. **Opportunity II: Configuring AutoML**

LLMs can suggest search spaces, budgets, fidelity settings, and other configuration choices by leveraging prior knowledge from similar tasks, reducing the need for expert manual setup.

3. **Opportunity III: LLMs as Components**
   LLMs can replace or augment AutoML modules such as candidate selection, surrogate modeling, and feature engineering, acting as meta-learned components that leverage broad prior knowledge.

2.4.3 State of the Art AI Agents for AutoML

Recent work investigates how agentic AI can reduce the complexity of AutoML and thereby increase its usability for non-expert users. Trirat, Jeong, and Hwang (2025) present a full-pipeline multi-agent framework that employs an agent manager to coordinate specialised agents for planning, data analysis, training-free model search and code generation. Their approach integrates retrieval-augmented planning, role-specific plan decomposition and multi-stage verification to produce correct and executable pipelines.

A complementary line of work focuses on improving the planning and search process itself. Chi et al. (2024) propose Tree-Search Enhanced LLM Agents (SELA), a agent-based system that leverages Monte Carlo Tree Search (MCTS) to optimize the AutoML process. Other approaches narrow their focus to specific parts of the pipeline: S. Liu, Gao, and Y. Li (2025) introduce AgentHPO, an LLM-driven agent for hyperparameter optimization, while T. Liu et al. (2024) propose Large Language Models to enhance Bayesian Optimization (LLAMBO), which leverages LLM to inject prior knowledge (such as promising configurations and constraints) into Bayesian optimization, thereby improving its sample efficiency.

Finally, case-based reasoning (CBR) has emerged as a promising mechanism for leveraging prior experience. Guo et al. (2024) demonstrate that CBR-based retrieval of similar past tasks enables agents to reuse model templates and hyperparameters, reducing search cost and improving stability in data science workflows.

Together, these approaches illustrate the rapid evolution of agentic AutoML, ranging from full-pipeline multi-agent systems to specialised planners, optimization agents and experience based retrieval methods.

2.4.4 Agentic AI Challenges

Despite rapid progress, current agentic AI systems face several limitations that constrain their reliability, robustness, and applicability. According to Krishnan (2025, pp. 37–39), existing agent benchmarks and architectures expose multiple technical challenges.

First, evaluation practices remain limited. Current benchmarks focus narrowly on accuracy while neglecting key metrics such as cost efficiency, reproducibility, robustness to distribution shifts, and

real-world applicability (Krishnan 2025, p. 2). This leads to agent designs that are unnecessarily complex and expensive, overfit to benchmark specifics, or exploit shortcuts rather than demonstrating genuine reasoning ability. In addition, the conflation of model-centric and downstream requirements makes it difficult to identify which architectures suit specific use cases (Kapoor et al. 2024, p. 1).

Second, as stated by Patil and Jadon (2025), several core architectural limitations persist. Modern agents struggle with reasoning depth, consistency, and verification. LLMs remain strong at linguistic fluency but weak in systematic and multi-step reasoning, often producing inconsistent or hallucinated intermediate steps. Context and memory management also remain a challenge, especially under long-horizon tasks requiring stable state tracking, planning, and temporal dependencies.

Third, tool use, planning, and integration capabilities remain immature. Agents frequently mismanage external tools, fail to recover from tool errors, and struggle to decompose goals into appropriate subtasks (Kapoor et al. 2024; Krishnan 2025).

Fourth, the use of LLMs for configuring AutoML introduces method-specific challenges. Extracting task relevant meta-knowledge often requires extensive prompt engineering, increasing human effort and reducing reproducibility. Moreover, the evaluation of LLM configured AutoML approaches on public benchmark datasets raises concerns about data leakage, as LLMs may have been exposed to these datasets during pretraining. Hallucinations further pose a critical risk, as incorrect assumptions or invalid configurations may be generated and executed automatically.

Fifth, the fluent natural-language interaction of LLMs can lead to overtrust by users, creating a misleading impression of reliability despite potentially flawed results. In addition, combining two resource-intensive research areas, namely LLMs and AutoML, substantially increases computational cost (A. Tornede et al. 2024, pp. 15–16).

Finally, scalability and computational efficiency pose practical constraints. High-quality agent performance is often achieved through repeated sampling, retries, or heavy reliance on large LLMs, inflating inference cost without improving underlying reasoning (Kapoor et al. 2024, p. 1–3). As a result, many reported accuracy improvements stem from increased compute budgets rather than better architectural design.

Chapter 3

Research Methodology

This chapter presents the research methodology underlying the proposed automated remaining useful life prediction system. It describes the problem formulation, system and software architecture, execution workflow, and the methodological choices that guide preprocessing, modeling, optimization, and dataset selection.

3.1 Problem Framing and Objectives

This thesis addresses the challenge of predicting the remaining useful life of technical systems. Traditional AutoML methods simplify model development but still rely on predefined search spaces and expert configuration, limiting their adaptability across diverse RUL datasets. Recent agentic AI approaches promise more flexible planning, dynamic search-space construction, and autonomous pipeline generation, yet their effectiveness for RUL prediction remains largely unexplored. This work evaluates three modeling strategies:

1. **AutoML standalone**: a classical AutoML approach based on a predefined search space.
2. **Hybrid (AutoML + agentic AI)**: a hybrid approach in which a multi-agent system designs the search space, while AutoML performs optimization.
3. **Standalone agentic AI**: an agentic approach in which a multi-agent system generates the complete modeling pipeline without AutoML.

The objective is to assess whether agentic AI can overcome key limitations of existing AutoML systems and improve automation, flexibility and modeling quality for RUL prediction. This leads to the following research questions:

- How do AutoML standalone, hybrid and standalone agentic AI approaches perform on standard benchmark RUL datasets?
- To what extent can agentic AI reduce manual configuration effort and increase adaptability across datasets?
- Can LLM-based code and pipeline generation serve as a practical complement or alternative to structured AutoML workflows?

3.2 System Architecture

This subsection outlines the conceptual architecture of the proposed system and provides a high-level overview of the workflow used to construct and evaluate RUL prediction models. The conceptual architecture is organized into two complementary systems: (i) the machine learning (ML) pipeline (Figure 3.1), which executes the data processing and model training steps, and (ii) the multi-agent system (Figure 3.2), which configures the entire pipeline and can propose new regression models. The ML pipeline is controlled via a YAML configuration file (see Listing B.1) that specifies preprocessing settings, feature engineering behavior, model definitions, and the associated hyperparameter search space.

3.2.1 Machine Learning Pipeline

The machine learning pipeline (Figure 3.1) takes raw time series sensor data as input and processes it through a sequence of preprocessing operations, including data cleaning, normalization and resampling.

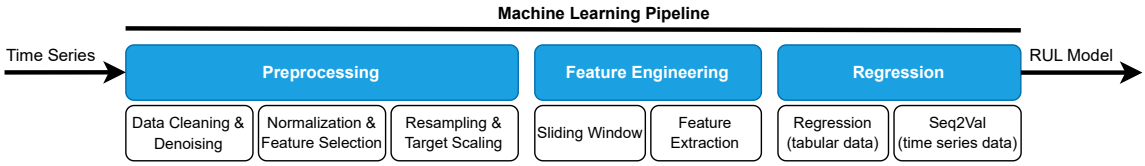


Figure 3.1: ML Pipeline for RUL Prediction

Depending on the modeling strategy, the pipeline either applies feature extraction to derive tabular representations for classical regression models or uses sequence-to-value architectures that operate directly on windowed time series data. The final stage predicts the RUL using either feature-based regression or sequence-based models.

3.2.2 Multi-Agent System

The multi-agent system (Figure 3.2) forms the configuration layer of the architecture. It follows a deterministic, rule-based workflow coordinated by the Manager Agent A_{mgr}, which sequentially invokes the Prompt Agent (A_p), Model Agent (A_m) and Config Agent (A_c).

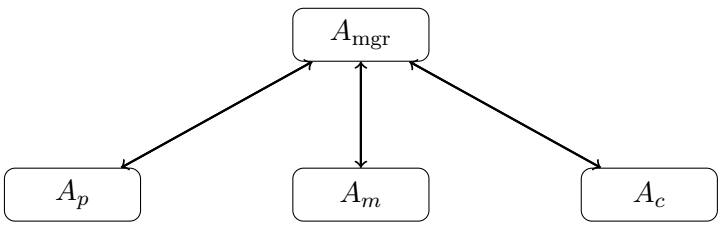


Figure 3.2: Multi-Agent System

The Prompt Agent collects user instructions, task constraints, dataset metadata and, when available, information from previous runs. It distils these inputs into actionable guidance for the downstream

agents. The Model Agent uses this information, together with the dataset schema, to propose new regression architectures that are not yet available in the model library. The Config Agent then constructs the machine learning pipeline configuration, including suitable preprocessing settings and an appropriate search space for the given dataset. Depending on the modeling mode, the resulting configuration is either combined with a predefined AutoML setup (hybrid approach) or fully determines the modeling pipeline in the agentic standalone setting.

3.2.3 Software Architecture

The software architecture is organized into three coordinated packages: `rul_lib`, `auto_rul`, and `agent_rul`. The `rul_lib` package provides core functionality for data handling, preprocessing, feature engineering, model training, evaluation, and supporting utilities shared across all execution modes. The `auto_rul` package implements the standalone AutoML workflow, including hyperparameter optimization. The `agent_rul` package realises the multi-agent system for automated pipeline configuration, reasoning, and model generation. Figure 3.3 illustrates the interaction between these packages and the surrounding infrastructure, including experiment tracking, model registry access, and monitoring components.

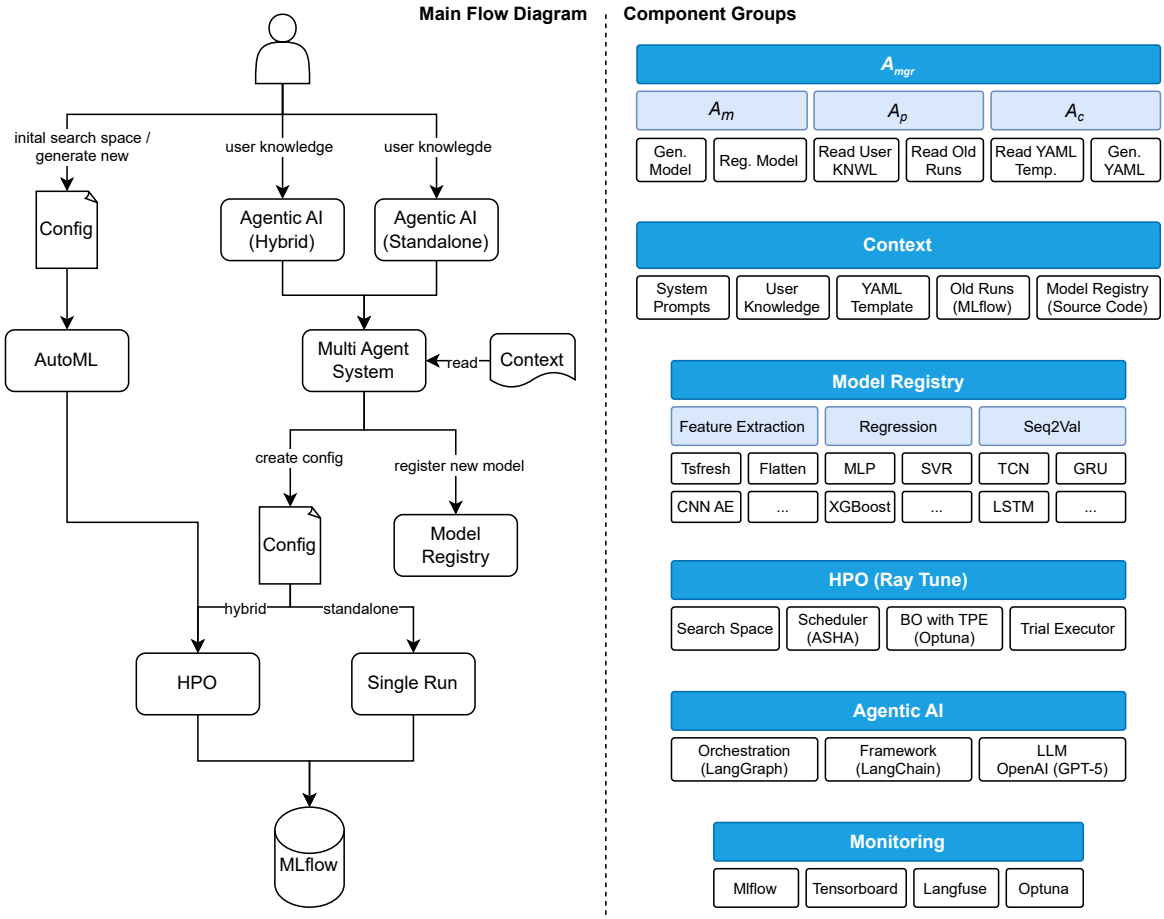


Figure 3.3: Software architecture of the AutoML and agentic RUL system.

Depending on the selected entrypoint, the system can be executed in AutoML, hybrid, or standalone agentic mode. The concrete execution semantics of these modes are described in Section 3.3.

3.3 Execution Workflow

Building on the software architecture described in Section 3.2.3, this section formalises the execution workflow of the system. The workflow specifies how the different software components are orchestrated at runtime, depending on the selected entrypoint.

Three execution modes are supported. In the *AutoML* mode, a predefined or user-provided configuration C_u is expanded into a search space and optimised using SMBO (subsection 2.3.5). In the *hybrid* mode, a multi-agent system generates a dataset-aware configuration C_a, which is subsequently optimised using the same AutoML backend. In the *standalone agentic* mode, the agent-generated configuration fully specifies the modeling pipeline and is executed without hyperparameter optimization.

Algorithm 2 summarises the resulting top-level execution logic and clarifies the interaction between user input, agentic configuration, and automated optimization.

Algorithm 2: Top-level execution workflow of the AutoML and agentic RUL system

Input: User knowledge K, dataset D, predefined or user-provided configuration C_u, entrypoint $e \in \{\textsc{AutoML}, \textsc{Hybrid}, \textsc{Standalone}\}$

Output: Trained model M, results R

`// The entrypoint` e `is selected by launching the corresponding program.`

if $e = \textit{AutoML}$ **then**

  Generate search space λ from C_u;

  $(R, M) \leftarrow \text{HPO}(\lambda)$; `// SMBO-based HPO`

else if $e = \textit{Hybrid}$ **then**

  $(C_a, \mathcal{M}) \leftarrow \textsc{InvokeMultiAgentSystem}(K, D, R_{\text{old}})$; `// Fig. 3.2`

  Generate search space λ from C_a;

  $(R, M) \leftarrow \text{HPO}(\lambda)$; `// SMBO-based HPO`

else

  $(C_a, \mathcal{M}) \leftarrow \textsc{InvokeMultiAgentSystem}(K, D, R_{\text{old}})$; `// Fig. 3.2`

  $(R, M) \leftarrow \textsc{RunSingle}(C_a)$; `// standalone agentic run`

Log R and M to MLflow;

`//` λ `specifies feature extraction, feature engineering, model class (feature-based or Seq2Val), and hyperparameters`

`//` $\mathcal{M}$ `denotes a model specification generated by the agent and registered in the model registry`

`//` $R_{\texttt{old}}$ `denotes previously logged results (if available), used by the Prompt Agent (`A_p`) for context-aware reasoning.`

3.4 Justification of Methods

This section summarises the rationale behind the methodological choices used in this work.

Supervised Regression
RUL estimation is naturally framed as a supervised regression task, as the goal is to predict a continuous lifetime value from multivariate sensor data. The window-based formulation provides a consistent input shape while capturing local temporal dynamics.

Search and optimization
Model quality depends strongly on preprocessing, feature selection and hyperparameter choices. Because these design spaces are large and dataset-dependent, automated search is more effective than manual tuning. The AutoML baseline relies on a predefined search space, while the hybrid approach uses an agent-generated configuration combined with HPO. In contrast, the standalone agentic approach directly proposes a concrete configuration and model architecture without hyperparameter optimization, enabling an explicit assessment of whether HPO remains necessary in an agentic setting.

Agentic AI
Agentic AI is used to decompose pipeline design into specialised roles (prompt analysis, model proposal and configuration synthesis). This enables dataset-aware reasoning and creates pipelines beyond fixed search spaces while keeping the workflow verifiable and deterministic.

3.5 Datasets

This section describes the datasets used for evaluation, including their characteristics and data structure.

3.5.1 C-MAPSS Turbofan Engine Dataset

The NASA C-MAPSS turbofan engine dataset (Arias Chao et al. 2021) is a simulated run-to-failure benchmark commonly used in prognostics and health management (PHM). It was generated using the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) tool and provides multivariate time series sensor measurements from aircraft engines operating under different fault and environmental conditions.

As shown in Table 3.1 the dataset is divided into four subsets (FD001–FD004), which differ in complexity based on the number of operating conditions and fault modes: FD001 contains a single fault mode and one operating condition, while FD004 combines multiple fault modes with multiple operating conditions, making it the most challenging.

Dataset	Operating Conditions	Fault Modes	Training Size	Test Size
FD001	1	1	100	100
FD002	6	1	260	259
FD003	1	2	100	100
FD004	6	2	248	249

Table 3.1: Overview of C-MAPSS, adapted from R. K. Gupta, Nakum, and P. Gupta (2025, p. 164)

Each engine trajectory runs until failure and is annotated with the remaining useful life, defined as the number of operational cycles before failure. As illustrated in Figure 3.4, most engines fail after about 200 cycles, and only a few trajectories extend beyond 300 cycles.

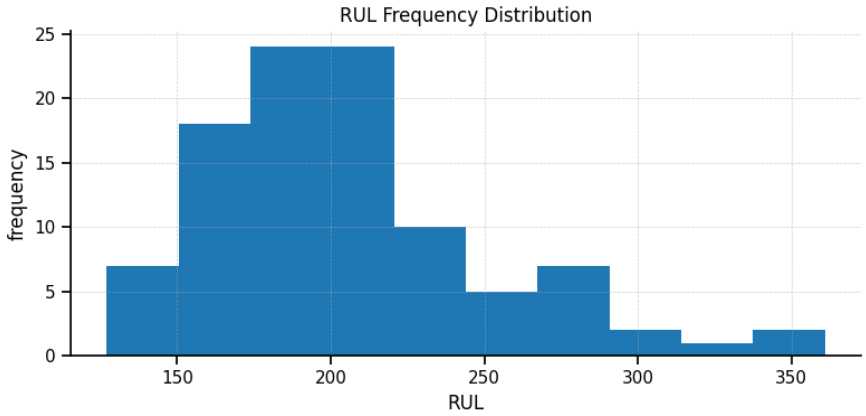


Figure 3.4: RUL frequency distribution (in cycles) over engines in FD001

Since degradation becomes observable only below roughly 125 cycles (Figure 3.5), the RUL is clipped at 125 to focus the model on the informative degradation region and to avoid learning from early-life samples where no deterioration is present (see Table A.1 in the appendix for sensor definitions).

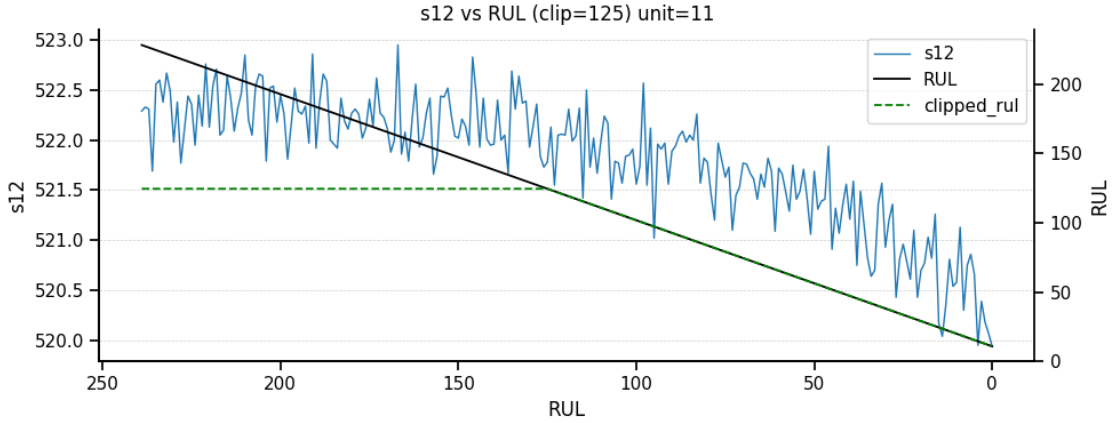


Figure 3.5: RUL clipping

The concrete application of the C-MAPSS dataset within the modeling pipeline is described in section 4.1.

3.5.2 FEMTO Bearing Dataset

The FEMTO-ST (PRONOSTIA) bearing dataset provides real run-to-failure measurements obtained from an accelerated degradation platform developed at the FEMTO-ST Institute (Nectoux et al. 2012). Vibration signals are recorded at a sampling frequency of 25.6 kHz, where each sample lasts 0.1 seconds and consists of 2560 data points recorded every 10 seconds. Unlike datasets where faults are artificially seeded, PRONOSTIA captures *naturally evolving* bearing degradation under controlled conditions, making it a representative benchmark for vibration-based prognostics. The platform consists of a rotating unit driven by an asynchronous motor, a pneumatic loading system applying radial forces up to 4000-5000 N, and a dedicated measurement module. Two orthogonally mounted accelerometers on the bearing housing record horizontal and vertical vibration.

Condition	Load (N)	Speed (RPM)	Experiments
Condition 1	4000	1800	7
Condition 2	4200	1650	7
Condition 3	5000	1500	3

Table 3.2: Operating conditions of the FEMTO (PRONOSTIA) experiments, reproduced from Dhungana, Rykkje, and Lundervold (2025).

Figure 3.6 illustrates the horizontal and vertical vibration signals for one run from Bearing 1 under Condition 1 (Table 3.2). As time progresses, the amplitude of the measured acceleration increases, reflecting the onset and growth of mechanical degradation.

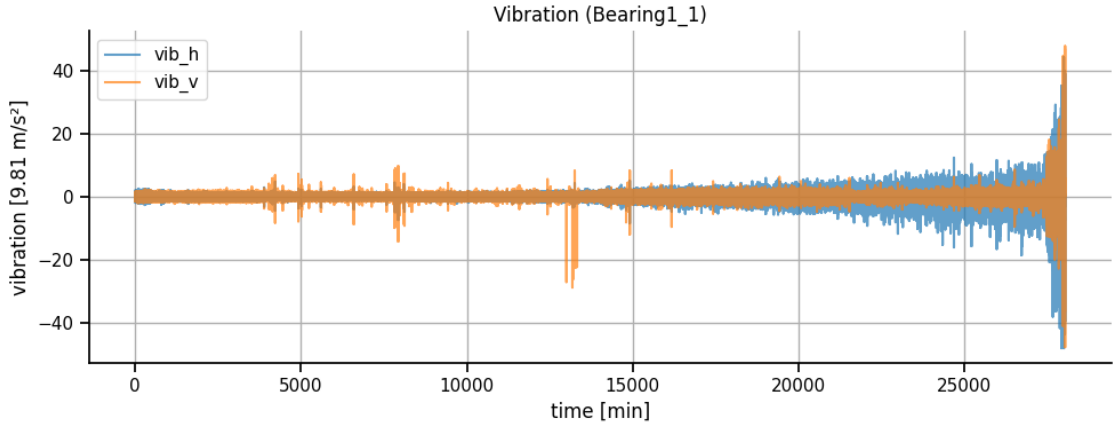


Figure 3.6: Horizontal and vertical vibration signals over time from Bearing 1 under Condition 1.

Spectrogram

To analyze how the frequency content evolves during degradation, the vibration data is also examined in the time–frequency domain. A spectrogram is obtained by applying the Fourier transform to overlapping windows of the signal. For each windowed segment $x_k(j)$ of length L, the power spectral density (PSD) is estimated from the Fourier transform $A_k(f)$ following the formulation of Welch

(1967):

$$I_k(f) = \frac{1}{L} \left| A_k(f) \right|^2, \qquad \hat{P}(f) = \frac{1}{K} \sum_{k=1}^{K} I_k(f), \tag{3.1}$$

where $I_k(f)$ denotes the periodogram of the k-th window and $\hat{P}(f)$ is the resulting PSD estimate, expressed in $(\mathrm{m/s^2})^2/\mathrm{Hz}$ and shown in logarithmic form (dB/Hz). Figure 3.7 shows the resulting spectrogram for a horizontal vibration channel. Early in the run the spectral energy remains low and stable; towards the end of life the PSD increases.

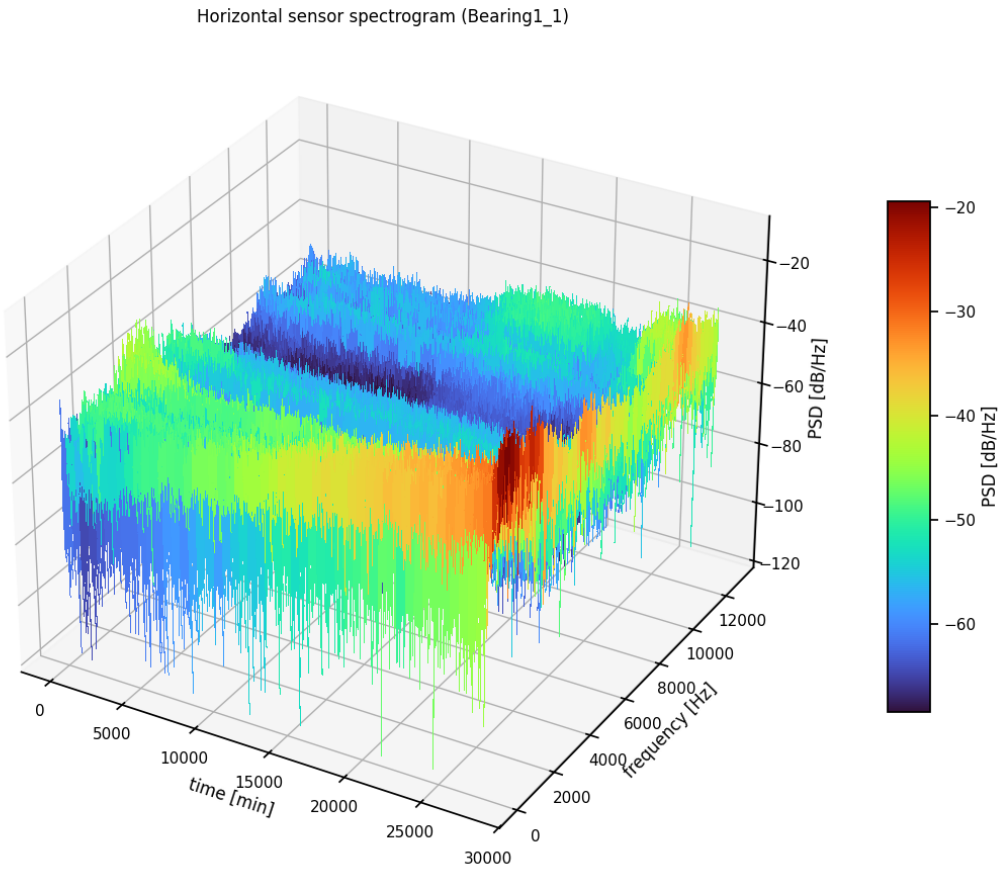


Figure 3.7: Spectrogram of horizontal vibration data from the FEMTO bearing dataset, computed using the Welch PSD estimator; implemented via Virtanen et al. (2020).

3.5.3 Dataset Suitability for the Research Question

The selected datasets are well aligned with the objectives of this work. Both C-MAPSS and FEMTO provide multivariate time series data capturing progressive run-to-failure degradation behavior, which is essential for evaluating remaining useful life prediction methods. In particular, C-MAPSS contains a large number of sensor channels describing different system states and operating conditions, enabling the study of remaining useful life estimation under high-dimensional input settings.

Both datasets are established public benchmarks in PdM research, which ensures comparability with existing approaches and supports reproducible evaluation. The FEMTO dataset complements

C-MAPSS by providing real-world bearing degradation data with high-frequency vibration measurements. It exhibits noise characteristics and contains only a limited number of run-to-failure trajectories, reflecting typical constraints encountered in industrial applications.

Using these two fundamentally different datasets supports the assessment of generalizability. They impose different requirements on preprocessing, feature engineering, and regression models, allowing the evaluation of whether the proposed RUL system can robustly handle heterogeneous data sources with varying signal characteristics, data volumes, and noise levels.

3.6 Preprocessing

Several preprocessing steps can be applied to prepare the raw multivariate time series for supervised RUL estimation. Implementation details and configuration examples are provided in chapter 4.

3.6.1 Data Cleaning and Denoising

Data quality has a direct impact on the stability of RUL models. The pipeline therefore supports two configurable steps: (i) imputation of missing values and (ii) optional denoising. In the agentic setting, the Prompt Agent (A_p) receives dataset metadata, including the dataset schema, and uses this information to select suitable cleaning operations.

Imputation

Missing values can be resolved using forward or backward filling, mean or median replacement, or zero filling. These approaches preserve the temporal structure of the sensor streams and prevent interruptions in the time series.

Denoising

Denoising improves feature stability under low signal-to-noise ratio (SNR) while remaining computationally efficient and preserving degradation patterns relevant for RUL prediction. The pipeline provides three lightweight options.

Exponential moving average (EMA): $y_t = \alpha x_t + (1-\alpha) y_{t-1},\ 0 < \alpha \leq 1$, where α controls the smoothing strength. EMA suppresses high-frequency noise while retaining slowly varying trends, which is well suited for vibration-like channels.

Moving average (MA): $y_t = \frac{1}{K} \sum_{i=0}^{K-1} x_{t-i}$, with window length K. This filter reduces random fluctuations and stabilises local segments, improving the consistency of downstream window extraction.

Time synchronous moving average (TSMA): generalises time domain synchronous average (TSA) by averaging only M neighbouring cycles of a periodic measurement $y(t) = x(t) + w(t)$ with period T. Whereas TSA forms a single fully averaged cycle, TSMA yields $N - M + 1$ partially

averaged cycles (Zhang and Hu 2019). Here, N denotes the total number of available cycles in the signal, and $M \leq N$ the number of neighbouring cycles used for averaging:

$$y_{\mathrm{TSMA}}(t) = \frac{1}{M} \sum_{m=0}^{M-1} y(t + mT) = x(t) + \frac{1}{M} \sum_{m=0}^{M-1} w(t + mT) \tag{3.2}$$

Because the underlying signal $x(t)$ is periodic, the averaging preserves its shape exactly while reducing the uncorrelated noise component. For $M = N$, TSMA reduces to TSA; for $M = 1$, it returns the raw measurement. Noise suppression is therefore weaker than TSA but stronger than a standard moving average, while retaining more high-frequency detail.

3.6.2 Normalization

As shown in Table 3.1, the C-MAPSS datasets contains multiple different operating conditions (e.g. altitude, Mach number). Sensor features therefore exhibit mode-dependent shifts and scale differences that are unrelated to degradation. Normalization mitigates these effects by putting features on a comparable scale and preventing the model from interpreting operating-regime variation as health related trends. This ensures that subsequent learning focuses on degradation behavior rather than raw magnitude differences imposed by the operating environment.

Normalization by Operational Condition

Let $x^{(m,d)}$ denote feature d observed under operating mode or cluster $m \in \{1, \ldots, K\}$. Operating regimes induce mode-dependent shifts and scales in sensor features that are *not* caused by degradation; mixing modes in a single scaler therefore biases learning toward regime differences instead of health-related variation. Following Peel (2008), standardization is performed within each operating mode:

$$N\left(x^{(m,d)}\right) = \frac{x^{(m,d)} - \mu_{(m,d)}}{\sigma_{(m,d)}}, \qquad \forall\, m, d \tag{3.3}$$

where $\mu_{(m,d)}$ and $\sigma_{(m,d)}$ denote the mean and standard deviation of feature d computed on training data restricted to mode m. This removes regime-specific nuisance effects, maximizes comparability within modes, and empirically improves RUL estimation accuracy.

Let $o = (o_1, \ldots, o_P)$ denote the operating-setting vector, where P is the number of operating-setting variables. Each sample is assigned a mode label $m \in \{0, \ldots, K\}$ for subsequent per-mode normalization, where K denotes the total number of distinct operating modes identified from the operating-setting vectors in the dataset. Depending on the characteristics of the operating settings, two cases are distinguished.

Few distinct rounded levels

Operating settings are first rounded,

$$r(o) = \mathrm{round}(o, \delta) \tag{3.4}$$

using a small step size (e.g. $\delta = 10^{-1}$). After rounding, let $\{u_m\}_{m=1}^{K}$ denote the set of distinct operating-setting vectors observed in the training data, where each u_m represents one operating mode. If the number of such distinct levels is small and each level has sufficient support, operating modes are assigned by exact matching:

$$m = g_{\text{unique}}(o) \iff r(o) = u_m \tag{3.5}$$

Unseen levels during inference: if a new sample does not match any known level, it is assigned to the closest one:

$$m^\star = \underset{m\in\{0,\dots,K\}}{\arg\min} \left\| r(o) - u_m \right\|_2 \tag{3.6}$$

Optionally, a tolerance τ may be used to fall back to global scaling when the closest level is too distant.

Selecting modes via clustering

When operating settings vary continuously, modes are obtained through clustering. The settings are first standardized:

$$\tilde{o} = S(o) = \frac{o - \mu}{\sigma} \tag{3.7}$$

For $K \in \{2, \dots, k_{\max}\}$, k-means clustering is fitted and evaluated using a combined internal validity score:

$$J(K) = 0.7\,\mathrm{CH}(K) + 0.3\,\widetilde{\mathrm{Sil}}(K) \tag{3.8}$$

where CH denotes the Calinski–Harabasz index and $\widetilde{\mathrm{Sil}} \in [0, 1]$ the normalized silhouette score. The selected number of clusters is

$$K^\star = \arg\max_K J(K) \tag{3.9}$$

Mode labels are assigned by nearest centroid:

$$m = g_{\text{kmeans}}(o) = \underset{1\le c\le K^\star}{\arg\min} \|S(o) - \mu_c\|_2. \tag{3.10}$$

where μ_c denotes the centroid of cluster c in standardized operating-setting space.

Inference for new samples

Mode assignment for unseen data follows the rule selected during training:

$$g(o) = \begin{cases} g_{\text{unique}}(o), & \text{if discrete levels were selected,} \\ g_{\text{kmeans}}(o), & \text{if clustering was selected.} \end{cases} \tag{3.11}$$

Notes: (i) $k_{\max}$ limits search time and prevents over-fragmentation. (ii) The resulting mode m is used to compute per-mode means and variances for sensor standardization.

3.6.3 Feature Selection

Feature selection is applied on the training split to reduce redundancy and retain only informative sensor channels. Two complementary strategies are supported: (i) correlation-based pruning and (ii)

mutual-information ranking. The selected feature set is then applied consistently to the validation and test splits.

Correlation-Based Pruning

Correlation-based pruning removes redundant features by analysing pairwise dependence between sensor channels. Let X denote the numeric feature matrix of x_{train}, excluding identifier or ignored columns. A dependence matrix is computed using the absolute Pearson correlation (see Equation B.2):

$$A_{ij} = |\text{corr}(X_i, X_j)| \tag{3.12}$$

where the absolute value is used since both positive and negative correlations indicate redundancy and contribute equally to multicollinearity.

To quantify the overall redundancy of each feature, a global dependence score is defined as

$$d_i = \frac{1}{p-1} \sum_{j \neq i} A_{ij} \tag{3.13}$$

which measures the average absolute correlation of feature i with all other features, with p denoting the total number of features. High values of d_i indicate strong redundancy, while low values suggest a more distinctive feature. This score approximates minimum redundancy by averaging pairwise feature dependencies (Hanchuan Peng, Fuhui Long, and Ding 2005). Given a correlation threshold τ, pruning is performed using a greedy procedure. Features are processed in ascending order of d_i. For the current feature i, all remaining features j with $A_{ij} \geq \tau$ are considered redundant and removed. This procedure iterates until no feature pairs exceed the threshold. By retaining features with low global dependence, the resulting subset preserves informative and complementary signals while eliminating multicollinearity. Similar correlation-based pruning strategies have been proposed in the literature; the presented approach provides a deterministic and computationally efficient variant based on global pairwise dependence.

Top k Selection by Mutual Information

A second strategy ranks features by their mutual information with the target variable (RUL). Let y denote the training target and $\text{MI}(X_i, y)$ its mutual information with feature X_i (see Equation B.1). Features are sorted by $\text{MI}(X_i, y)$ in descending order, and the top k features are retained. This approach selects features that carry the strongest predictive signal with respect to the target.

Application Across Splits

Both methods are fitted exclusively on `x_train`. The selected feature set is then applied to `x_test` to ensure consistency. Metadata stored in `meta` includes the chosen method, thresholds, selected features, and diagnostic statistics (dependence scores or mutual information values).

3.6.4 Resampling and Target Scaling

Before training, the pipeline optionally applies two preprocessing steps: *resampling* followed by *target scaling*. Resampling reduces the dominance of healthy samples, and target scaling ensures that the target values lie in a numerically stable range for regression models. Both components are fully configurable through the preprocessing settings (section B.1) of the ML pipeline.

Resampling

The input data may be resampled to reduce the imbalance between healthy and degrading operating regions. The procedure keeps all samples whose RUL is below the clip threshold and retains only a configurable fraction of the remaining healthy samples per unit. The strategy supports both row-level and cycle-level resampling and can be enabled or disabled independently for the training and test splits. Resampling parameters, dataset statistics (e.g. the numbers of retained healthy and degrading samples), and learned preprocessing parameters (e.g. normalization constants) are recorded in the pipeline metadata for reproducibility and consistent inference (see Listing B.2). This ensures that the same preprocessing transformations learned on the training data are applied during inference.

Target Scaling

After resampling, the target values (RUL) are clipped at a specified upper limit and then transformed using a configurable scaling method. The pipeline supports standardization, min-max scaling, or no scaling. Only the designated target column is modified, and all scaling parameters are derived exclusively from the resampled training split to ensure consistent application to validation and test data. The chosen method, target column, clip value, and fitted parameters are stored in `meta[y_scaler]`.

3.7 Feature Engineering

Feature engineering prepares the preprocessed multivariate time series for learning by transforming variable length degradation trajectories into fixed- size representations. This is achieved through sliding-window segmentation, followed by optional feature extraction, enabling both classical regression models and Seq2Val deep learning architectures to be applied in a unified pipeline.

3.7.1 Sliding Window

To prepare the sensor data for sequence models, each degradation trajectory is treated as a MTS and divided into fixed-length windows. Following the definition of a MTS by Darban et al. (2025, pp. 3–4), a time series with T time points can be written as $X = (X_1, X_2, \ldots, X_T)$ where each vector $X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,d})$ contains the d sensor observations at time step i. A *sliding window* of length L extracts a contiguous subsequence,

$$W_t = (X_t, X_{t+1}, \ldots, X_{t+L-1}), \qquad t = 1, \ldots, T - L + 1 \tag{3.14}$$

and successive windows are spaced by a configurable *stride* S. When $S < L$, windows overlap. Each window receives the RUL value at its final time step and serves as one training example for the Seq2Val models.

Starting from a time series of dimension $(T \times d)$, the sliding-window procedure produces a stack of fixed-length segments of shape $(N_{\text{win}} \times L \times d)$, each paired with its corresponding RUL label.

3.7.2 Multi Window

Sliding windows provide local snapshots of the degradation process. However, a single window of length L may not capture sufficient temporal context, especially when degradation patterns evolve slowly or when high-frequency data requires short window lengths. To incorporate a longer history without increasing L, this work employs a *multi-window* representation that aggregates information from multiple consecutive windows.

Let $W_t \in \mathbb{R}^{L \times d}$ denote the sliding window starting at time index t, as defined in subsection 3.7.1. For a fixed number of consecutive windows $M \geq 1$, the multi-window input is constructed as:

$$\widetilde{W}_t = (W_t, W_{t+1}, \ldots, W_{t+(M-1)}) \tag{3.15}$$

In practice, this representation can be utilised in two different ways, depending on the downstream model.

Feature or latent stacking for tabular regression
Given a feature extractor $\phi(\cdot)$ that maps a window to a feature or latent representation $f_t = \phi(W_t) \in \mathbb{R}^F$ (e.g. tsfresh features, see subsection 3.7.3 or an autoencoder embedding), the multi-window feature vector is obtained by concatenation

$$\widetilde{f}_t = [f_t \parallel f_{t+1} \parallel \ \ldots \ \parallel f_{t+(M-1)}] \in \mathbb{R}^{M \cdot F} \tag{3.16}$$

This enables classical regressors (e.g. gradient-boosting methods or multilayer perceptrons (MLPs)) to exploit temporal context spanning M windows while operating on a fixed-size tabular input.

Window stacking for Seq2Val models.
For sequence-based models, the M consecutive windows are concatenated along the temporal axis, yielding a stacked input tensor

$$\widetilde{X}_t \in \mathbb{R}^{(M \cdot L) \times d}. \tag{3.17}$$

This increases the effective temporal receptive field of the model without changing the per-window resolution or sampling frequency.

In both cases, the target label is assigned consistently with the sliding-window setup, using the remaining useful life value at the final time step of the last window in the multi-window block. The number of aggregated windows M is treated as a configurable parameter, allowing the pipeline to

trade off temporal context against input dimensionality and computational cost.

3.7.3 Feature Extraction

After windowing, each window is a multivariate time series of shape $(L \times d)$. Feature extraction converts these windows into tabular feature vectors. In general, feature engineering refers to transforming raw inputs into representations that improve the performance of a learning algorithm (Hollmann, Müller, and Hutter 2023, p. 3). Models that require tabular inputs, such as linear models, tree-based ensembles, gradient-boosting methods, and MLP, use this representation to obtain a feature matrix of shape $(N_{\text{win}} \times F)$, where F denotes the number of extracted features. Seq2Val deep learning models instead operate directly on the window tensor $(N_{\text{win}} \times L \times d)$ and therefore do not require explicit feature extraction.

The pipeline supports several strategies:

- **Tsfresh**: automatically computes a comprehensive set of statistical and signal-based features for each window, combining more than sixty underlying characterization methods into several hundred features by default (Christ et al. 2018). Tsfresh also performs automated relevance filtering, enabling the extraction of meaningful time series characteristics without manual feature engineering.
- **Flattening**: reshapes the window $W_t \in \mathbb{R}^{L \times d}$ into a single vector of length $L \cdot d$. This preserves all raw sensor values but no features are learned with this approach.
- **CNN Autoencoder**: learns a compact latent representation $z_t \in \mathbb{R}^k$, where $k < L \cdot d$, capturing temporal structure and spatial relationships in an unsupervised manner.

For models using multi-window representations, the extracted features or latent vectors are aggregated as described in subsection 3.7.2.

3.8 Regression

This section outlines the two modeling families used in the pipeline for predicting remaining useful life: regression methods that operate on tabular representations, and Seq2Val models that learn directly from windowed multivariate time series.

3.8.1 Feature-Based Regression

Feature-based regression methods take a fixed-length vector as input and apply a supervised learning model to estimate the RUL. Examples include linear models, tree-based ensembles, gradient-boosting methods, and multilayer perceptron, though any tabular regression model can be used. These models do not process temporal sequences directly and therefore rely on the tabular representations produced earlier in the pipeline. When multi-window representations are used, features or latent vectors from multiple consecutive windows are concatenated into a single input vector.

3.8.2 Seq2Val

Seq2Val models ingest the full window ($L \times d$), learning temporal and spatial dependencies end-to-end. Typical architectures include long short-term memory (LSTM), gated recurrent unit (GRU), and temporal convolutional network (TCN), but the pipeline is compatible with any sequence model operating on windowed multivariate time series. When multi-window representations are applied, consecutive windows are concatenated along the temporal axis, increasing the effective sequence length. By using the raw sequences directly, these models capture degradation dynamics without requiring a tabular intermediate representation.

3.9 Training

The training procedure depends on the selected regression model. Classical feature-based models (e.g. Random Forests, Support Vector Regression, Gradient Boosting) rely on model-specific fitting procedures, whereas neural network models require gradient-based optimization over multiple epochs. To enable a unified optimization interface, all model families are evaluated using a common validation loss during hyperparameter optimization.

Loss Function

To enable a unified optimization interface, all model families expose a common validation loss $\mathcal{L}_{\text{obj}}$ for hyperparameter search. By default, this objective is the mean squared error (MSE):

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2 \tag{3.18}$$

Depending on the optimization setting, alternative objectives can be selected, including the NASA scoring function (Equation D.2), the Huber loss for improved robustness to outliers, and a combined objective $\mathcal{L}_{\text{MSE}} + \lambda \mathcal{L}_{\text{NASA}}$ with a configurable weighting factor λ. Independent of the chosen objective, models are reported using standard regression metrics such as MSE and RMSE. While RMSE is reported for interpretability, MSE is typically used as the optimization loss due to its smoother gradients and computational convenience.

Training of Classical Models

Feature-based models such as Random Forests or Gradient Boosting models are trained using their native fit procedures as implemented in scikit-learn (Pedregosa et al. 2011). These models do not iterate over epochs. Instead, their hyperparameters (e.g. number of trees, maximum depth, learning rate) control model complexity. During optimization, the validation MSE obtained after a single fit call is returned to the scheduler.

Training of Neural Models

Neural models (e.g. convolutional neural network (CNN), LSTM, TCN) are trained using mini-batch gradient descent. Unless restricted by the search space, all models use the Adam optimizer as it

combines momentum and RMSprop-style adaptive learning rates and therefore serves as a robust baseline. A training epoch consists of a full pass over the training set followed by computation of the validation loss.

Fidelity and Early Termination

For neural models, the number of training epochs serves as the fidelity level in the multi-fidelity asynchronous successive halving algorithm (ASHA) scheduler. Promising configurations are allocated additional epochs, while weak ones are terminated early based on intermediate validation losses. Classical models do not support progressive training; they implicitly operate at a single fidelity level.

Training Output

Each trained model (classical or neural) returns its validation loss and the final fitted estimator. These outputs are consumed by the optimization procedure or, in the standalone agentic mode, directly used for downstream evaluation.

3.10 Multi-Agent System Components

This section describes the components of the multi-agent system presented in subsection 3.2.2 in more detail.

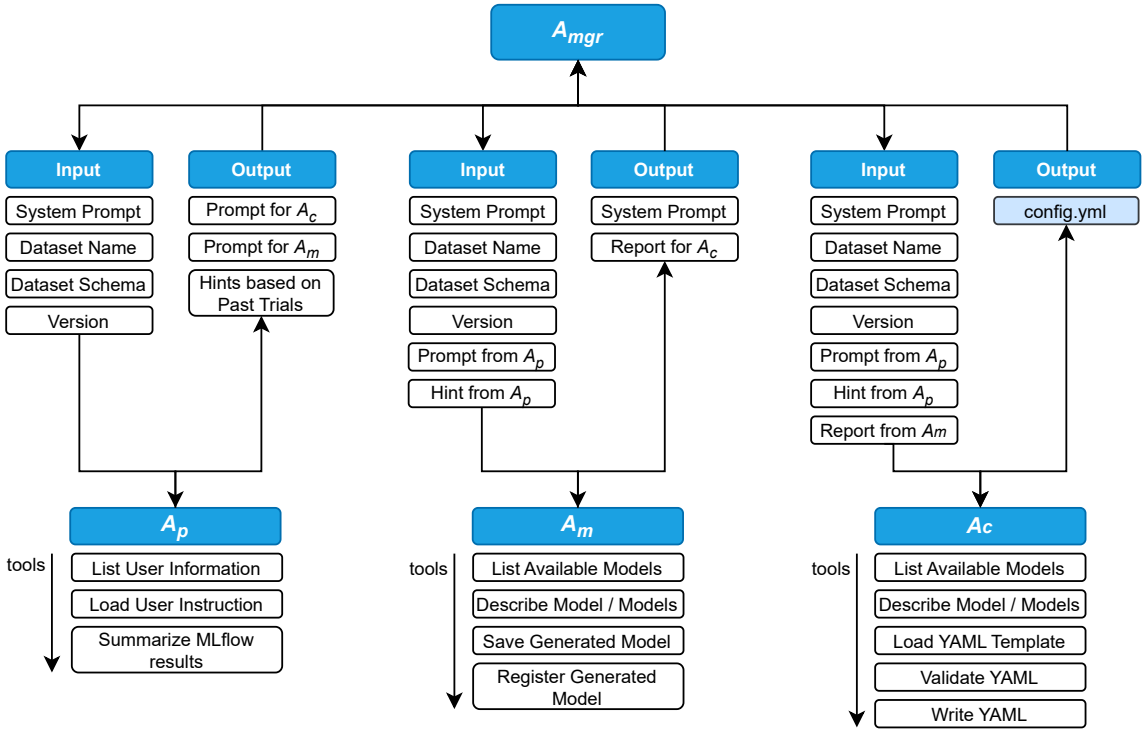


Figure 3.8: Multi-Agent Architecture

As one can see in Figure 3.8, instead of having a single agent responsible for all tasks, the system is decomposed into four specialised agents, each focusing on a specific aspect of the ML pipeline.

The multi-agent system is implemented using LangChain (LangChain 2026a) for agent abstraction and tool integration, with LangGraph (LangChain 2026b) providing graph-based orchestration of agent execution. Reasoning and code-generation capabilities are realized through a large language model accessed via the OpenAI interface (OpenAI 2025) using GPT-5. The execution is manager-centred: the Manager Agent initiates the process, invokes each specialised agent when needed, and always regains control after their completion. The invocation pattern therefore follows a hierarchical structure.

Manager Agent (A_{mgr})

The Manager Agent serves as the central coordinator of the multi-agent system and acts as the main interface between the specialised LLM agents and the underlying ML pipeline. It orchestrates the overall workflow by invoking the Prompt, Model, and Config Agents and by consistently regaining control after each agent call.

The agent receives experiment metadata, such as dataset identifiers and version information, together with the dataset schema. Before executing the workflow, it checks whether a cached configuration already exists and returns it if available. If no cached configuration is found, the Manager Agent initializes the shared agent state and iteratively invokes the specialised agents until a valid pipeline configuration has been produced.

In the current implementation, routing follows a deterministic sequence ($A_p \rightarrow A_m \rightarrow A_c \rightarrow$ Finish). However, the infrastructure is designed to support an LLM-based routing mechanism that could enable adaptive decision-making in future extensions. The output of the Manager Agent is the final YAML configuration used to execute the ML pipeline.

Prompt Agent (A_p)

The Prompt Agent transforms dataset metadata, user feedback, and historical experiment results into compact, machine-readable guidance for downstream agents. Its primary function is to synthesise high-level modeling instructions by combining the dataset schema (Table A.2), user-provided notes, and summaries of previous runs.

Based on this information, the agent analyzes feedback, summarises past experiments, identifies performance trends, and derives concise modeling guidance for the Model and Config Agents. The Prompt Agent relies on LLM-based reasoning and has access to tools for reading markdown feedback and querying MLflow experiment summaries. Its output is a single, strictly JSON-formatted object encoding modeling hints, search-space cues, and model preferences.

Model Agent (A_m)

The Model Agent designs and registers suitable RUL regression models based on the dataset, its schema, and the guidance provided by the Prompt Agent. Its primary responsibility is to translate high-level modeling hints into concrete model implementations, covering both feature-based (tabular regression) and Seq2Val architectures.

The agent receives as input the dataset and its schema, a list of available model families and recommendations from the Prompt Agent.

Based on this information, the Model Agent generates Python modules that implement a top-level model class. It then invokes dedicated tools to persist the generated module (write the generated Python module to disk) and to register the corresponding model adapters (tabular or Seq2Val) in the model registry.

The Model Agent relies on LLM-based code synthesis and tool usage for model inspection, persistence, and adapter registration. Its output consists of the saved and registered model modules, together with a structured model-creation report that is returned to the Manager Agent.

Config Agent (A_c)

The Config Agent translates the guidance produced by the Prompt and Model Agents into an executable YAML configuration for the RUL AutoML pipeline. It is responsible for constructing and validating a complete pipeline specification, including preprocessing steps, feature engineering, feature extraction methods, regression models, and their associated hyperparameter search spaces.

The agent receives the dataset schema, a YAML template path, contextual hints from the Prompt Agent (e.g. regarding cleaning, normalization, and resampling), the Model Agent's report containing proposed models, and a `standalone` flag that determines whether a single fixed model or a full HPO search space should be used.

Using this information, the Config Agent queries available models and schedulers, fills the YAML template, fixes preprocessing parameters outside the search space, and designs search spaces for feature extraction, regression and Seq2Val components. When `standalone` is enabled, the agent configures exactly one model without hyperparameter optimization. Otherwise, it includes all models proposed by the Model Agent, together with compatible models from the predefined model registry, and defines appropriate search spaces for each.

The generated configuration is validated using a lightweight syntactic and structural check. Validation ensures valid YAML format and the presence of required top-level sections (`dataset`, `preprocessing`, `feature_engineering`, `regression`). The validated configuration is then written to disk and returned to the Manager Agent in structured form

3.11 Optimization

The predictive performance of RUL models depends on several design choices related to feature engineering (section 3.7), regression model selection (section 3.8) and hyperparameters. In contrast, preprocessing operations are *not* included in the optimization process. They are applied deterministically based on the given configuration. Excluding preprocessing from the search space avoids excessive runtime, prevents combinatorial growth of the optimization problem, and improves the reproducibility of the pipeline.

3.11.1 AutoML Standalone

In the AutoML standalone approach, hyperparameter optimization is carried out using Optuna (Akiba et al. 2019) together with Ray Tune (Liaw et al. 2018). Optuna provides the sequential model-based optimization mechanism (see subsection 2.3.5) via its Tree-structured Parzen Estimator (TPE) sampler (Bergstra et al. 2011, p. 3), which serves as a surrogate model for proposing new configurations.

Tree-structured Parzen Estimator

The TPE algorithm (Bergstra et al. 2011) constructs a probabilistic model over promising and unpromising regions of the hyperparameter space and uses this model to select new configurations. The key idea is to express the surrogate model in terms of the conditional density $p(x \mid y)$ and the marginal $p(y)$, where x denotes a hyperparameter configuration and y its observed validation loss.

All previously evaluated trials are split at a quantile threshold γ into a set of *good* configurations with losses $y < y^*$ and a set of *bad* configurations with losses $y \geq y^*$, where y^* is the γ-quantile of the observed losses. Two non-parametric density estimators are then fitted:

$$p(x \mid y) = \begin{cases} \ell(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases} \tag{3.19}$$

(Bergstra et al. 2011, p. 4) with $\ell(x)$ representing the density of hyperparameters associated with low losses and $g(x)$ representing those associated with higher losses. This formulation enables an analytic expression for the expected improvement acquisition function. Since $p(x) = \gamma\,\ell(x) + (1-\gamma)\,g(x)$, the expected improvement (EI) for a candidate x becomes

$$EI(x) \propto \left(\gamma + (1-\gamma)\,\frac{g(x)}{\ell(x)}\right)^{-1} \tag{3.20}$$

Thus, improvement is maximised for configurations that are likely under $\ell(x)$ and unlikely under $g(x)$.

Orchestration of Trials

Having defined the surrogate model and acquisition strategy, the remaining AutoML procedure concerns the orchestration and execution of trials. Each trial corresponds to a complete instantiation of the machine learning pipeline, including feature extraction, model training, and validation.

Trial execution is orchestrated using Ray Tune, which schedules multiple trials in parallel across the available compute resources. To improve computational efficiency, multi-fidelity optimization is employed via an ASHA scheduler (Liam Li et al. 2020). ASHA evaluates configurations using intermediate validation results and terminates underperforming trials early, thereby allocating additional resources to promising configurations.

In the proposed pipeline, the fidelity dimension corresponds to the number of training iterations (epochs). Configurations that achieve favourable validation performance are allowed to continue training with increased budgets, while poorly performing ones are pruned at early stages. This strategy reduces overall runtime while preserving competitive performance.

To avoid redundant computation across trials, expensive intermediate results, such as extracted feature matrices, are cached using Ray's distributed object store. This shared feature cache enables multiple trials to reuse preprocessing and feature extraction outputs while still exploring different models and hyperparameter settings.

The optimization objective is to minimise the validation mean squared error (Equation 3.18). The search process terminates when either a predefined time budget or a maximum number of trials is reached.

3.11.2 Hybrid: Agentic AI and AutoML Approach

In the hybrid setting, the multi-agent system (subsection 3.2.2) provides a data- and user instruction aware configuration before optimization begins. AutoML then applies SMBO within this agent-generated search space. While the optimization mechanism remains identical to the baseline, the search space is no longer fixed but adapted to the dataset characteristics. This improves efficiency and reduces the need for manually engineered model search spaces, while still ensuring that preprocessing is executed only once outside the optimization loop.

3.11.3 Standalone Agentic Approach

The standalone agentic mode removes the hyperparameter optimization loop entirely. Instead, the agent system generates the complete modeling pipeline, including preprocessing decisions, feature extraction strategy, and regression model configuration. No SMBO or multi-fidelity scheduling is applied. The resulting pipeline is thus the outcome of agent reasoning rather than iterative search, prioritising autonomy and rapid model generation. Since preprocessing and model settings are produced directly by the agents, no optimization resources are spent on repeated training-evaluation

cycles.

3.11.4 Summary

The three modeling modes primarily differ in how optimization and model design are handled within the pipeline. In the AutoML baseline, a predefined set of models and a fixed search space are explored using SMBO, with optimization restricted to model-related hyperparameters and preprocessing applied outside the loop. The hybrid approach retains the same optimization mechanism but replaces the static search space with a dataset-aware, agent-generated configuration, which also introduce newly generated model architectures. In contrast, the standalone agentic approach removes iterative optimization entirely and relies on agent reasoning to generate both the complete pipeline and the regression model.

Chapter 4

Implementation

At the time of writing, the implementation is maintained in a private GitLab repository and can be made available to examiners upon request.

4.1 C-MAPSS Example

This section illustrates how the system is applied to the C-MAPSS dataset introduced in subsection 3.5.1, using the hybrid agentic AI approach.

4.1.1 Agentic Reasoning Process

For the C-MAPSS benchmark, the Prompt Agent (A_{p}) analyzes dataset characteristics and, when available, prior optimization results to derive structured guidance for the downstream agents. An example user instruction for FD002 is shown in Listing C.2.

Based on this guidance (Listing C.5), the Model Agent (A_{m}) proposes suitable model candidates, including a dataset-specific Seq2Val architecture (Listing C.6). The Config Agent (A_{c}) with guidance from A_{p} (Listing C.8) then synthesises the final pipeline specification, either by defining a hyperparameter search space in the hybrid setting or by producing a single configuration in the standalone agentic setting.

The resulting configuration is thus explicitly tailored to the characteristics of the FD002 subset. A complete configuration example is provided in section C.2.

4.1.2 Preprocessing

Normalization

As discussed in subsection 3.6.2, normalization is needed to handle mode-dependent shifts and scale differences in the operating conditions and sensor signals. For the FD002 subset, clustering-based normalization was applied with a maximum of six operating modes. The operating-condition features form six well-separated clusters (Figure 4.1), and the settings vary only slightly within each mode.

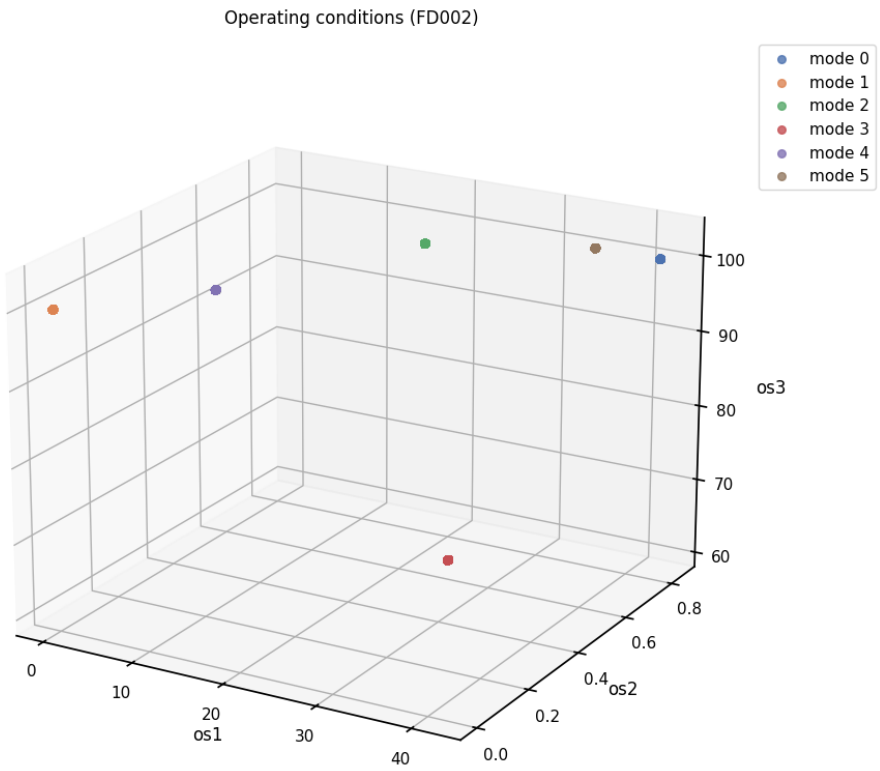


Figure 4.1: Operating conditions (FD002)

Before normalization, the sensor distributions differ substantially across the detected modes (Figure 4.2).

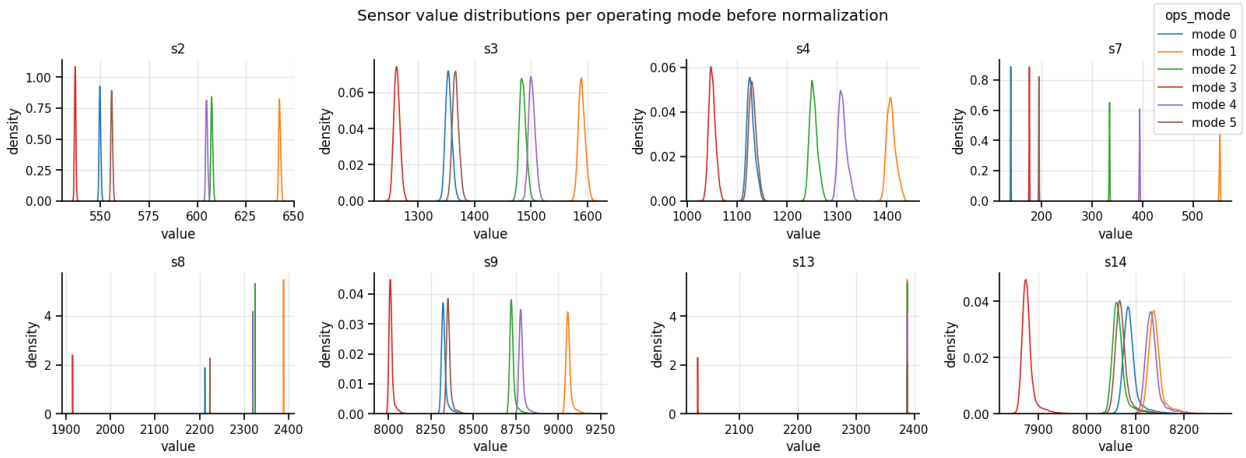


Figure 4.2: Sensor distributions per operating mode (FD002)

After applying clustering-based normalization, the sensor distributions align well across modes (Figure 4.3). This illustrates that the normalization strategy effectively removes mode-specific offsets and scale differences.

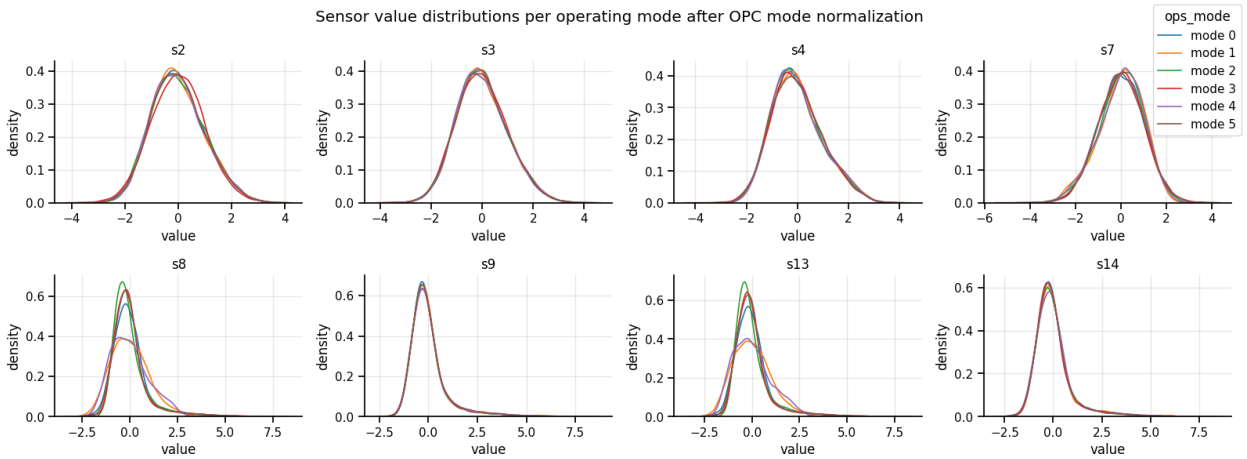
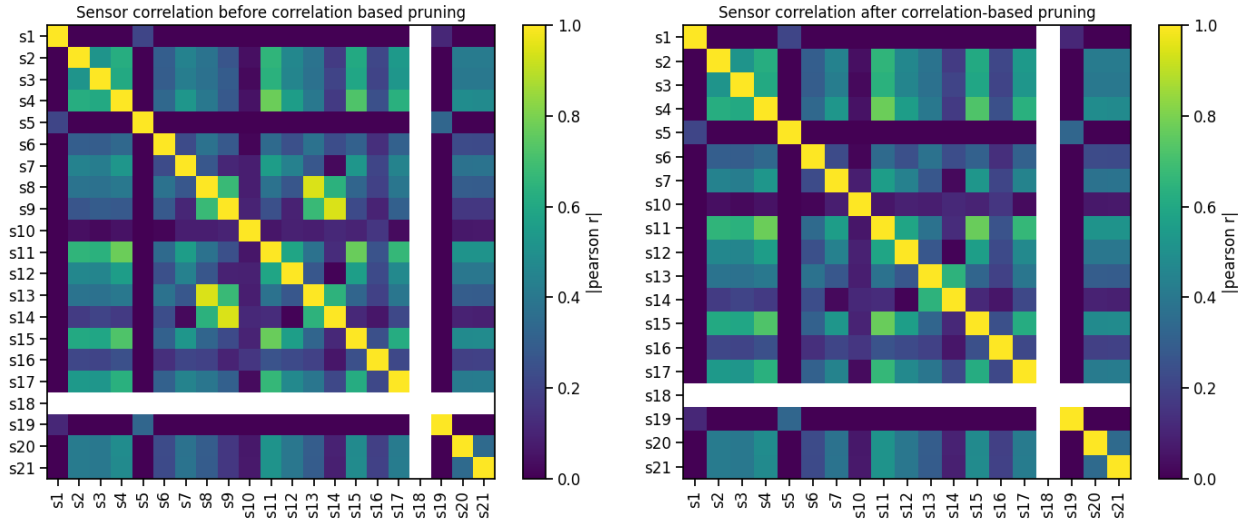


Figure 4.3: Sensor distributions per operating mode (FD002)

Feature Selection

To reduce redundancy and improve model efficiency, correlation-based feature selection was applied as described in section 3.6.3. Figure 4.4a shows the dependence matrix of the normalized sensor signals. Several sensors exhibit high pairwise correlations, indicating redundant measurements. Applying a threshold of $\tau = 0.9$ yields the reduced set of features shown in Figure 4.4b.



(a) Dependence matrix before pruning

(b) Dependence matrix after pruning

Figure 4.4: Dependence matrices for sub-dataset FD002

Sensors `s8` and `s9` were removed because they showed strong correlation with `s13` and `s14`, respectively. After mode-aware normalization, `s18` became constant, and therefore no meaningful correlation could be computed for this channel. Such variance-free features are discarded automatically by the mutual information based top-k selection (section 3.6.3), since they carry no information about the RUL target. This behavior is visible in the mutual information ranking shown in Figure 4.5, where only the top-k features (here with $k = 12$) are selected for modeling.

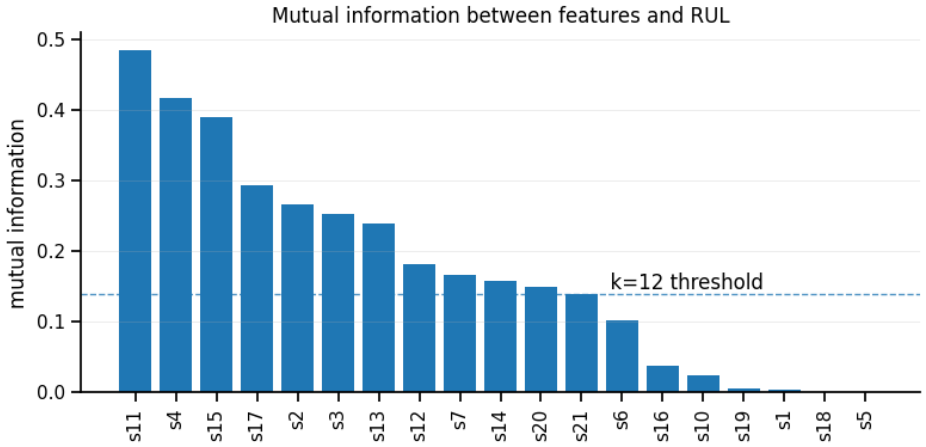
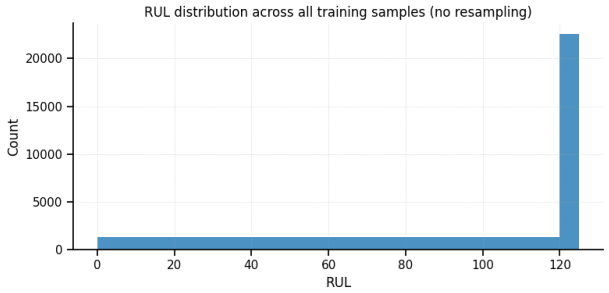


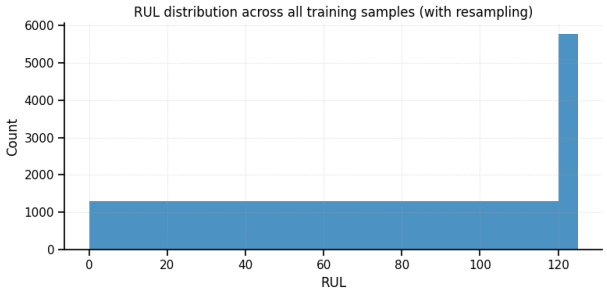
Figure 4.5: Mutual information between features and the RUL target (FD002)

Resampling

As stated in subsection 3.6.4 resampling can be applied to mitigate imbalanced RUL distributions between healthy (RUL ≥ 125) and degrading (RUL < 125) states.



(a) RUL distribution before resampling



(b) RUL distribution after resampling

Figure 4.6: RUL distributions for sub-dataset FD002

Figure 4.6 shows that the applied strategy reduces the dominance of healthy samples by selectively downsampling healthy trajectories, yielding a more balanced distribution that improves model training stability.

4.1.3 Feature Engineering

The C-MAPSS sensor trajectories are transformed into fixed-length sequences using the sliding-window procedure introduced in subsection 3.7.1. In the implementation, each unit's multivariate time series is segmented into windows of length L with stride S, and each window receives the RUL label at its final time step. Figure 4.7 illustrates the sliding-window extraction for sensors `s2` and `s14` of unit 2 in subset FD002.

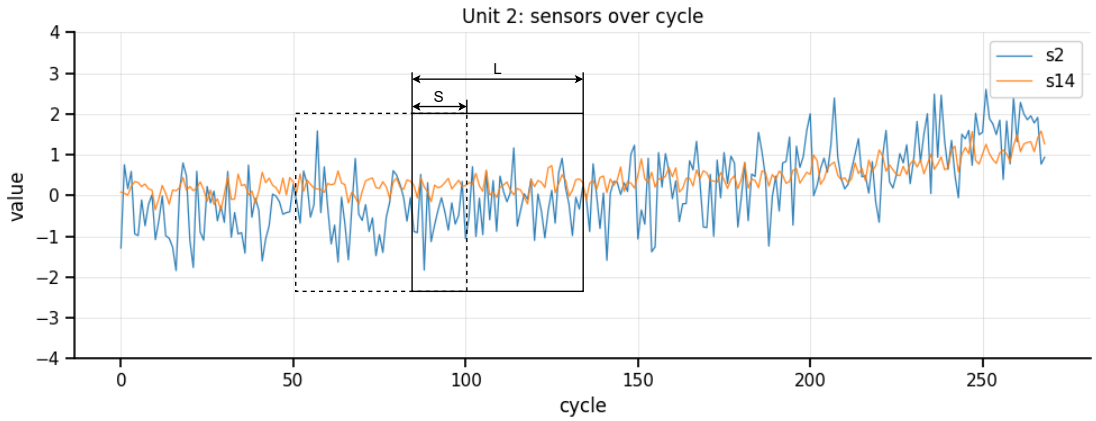


Figure 4.7: Sliding-window extraction with window length L and stride S (FD002).

The resulting dataset has shape $(N_{\text{win}} \times L \times d)$, where N_{win} depends on both L and S. In the hybrid run on FD002, the automatically selected model is a GRU (subsection 4.1.4). No explicit feature extraction is required, since the sequence model directly consumes the raw windowed time series input.

4.1.4 Training and Optimization

In this experiment, five representative validation trajectories were selected for detailed analysis, focusing on the best-performing operating mode. The Model Agent generated a Seq2Val GRU architecture (Listing C.6) by extending the standard GRU implementation from the model registry with optional bidirectional recurrence. In a bidirectional GRU, the input sequence is processed both forward and backward in time, and the hidden states from both directions are combined. This allows the model to exploit past and future temporal context within each sliding window, which can be beneficial when degradation patterns are not strictly monotonic within the window. Despite this architectural extension, the generated model did not outperform the predefined GRU variants already available in the registry, although comparable validation performance was achieved.

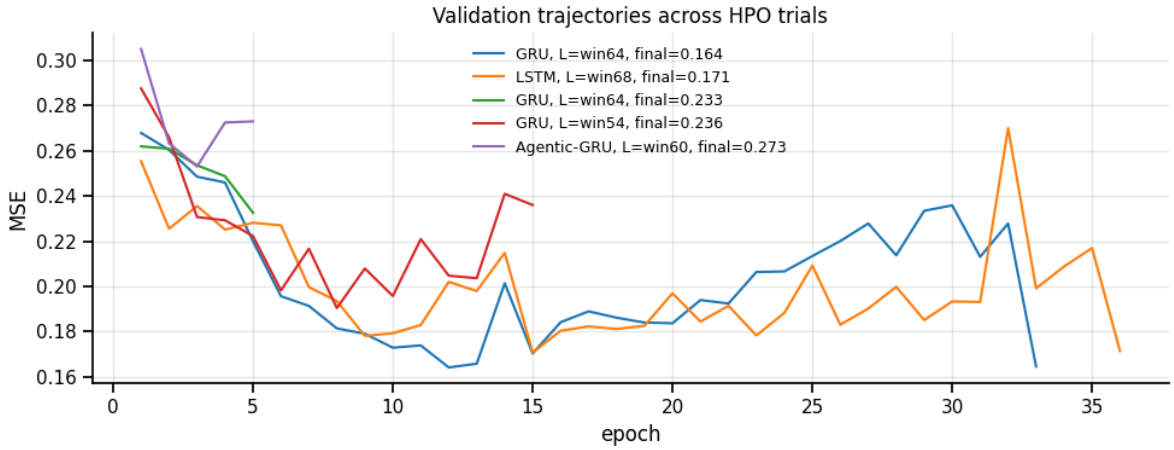


Figure 4.8: HPO trials on validation trajectories for FD002.

As shown in Figure 4.8, the HPO process explores diverse hyperparameter configurations, including different Seq2Val architectures and window lengths. Trial execution and early termination follow the multi-fidelity ASHA strategy described in section 3.11.1. The search converges towards low validation error, with the best configuration achieving a normalized validation MSE of approximately 0.16.

To further analyze the influence of temporal context, Figure 4.9 visualises the HPO response surface with respect to window length and Seq2Val model architecture. Window length has a pronounced impact on performance. The optimal region lies between roughly 60 and 70 time steps, which aligns well with the automatically selected window length of $L = 64$.

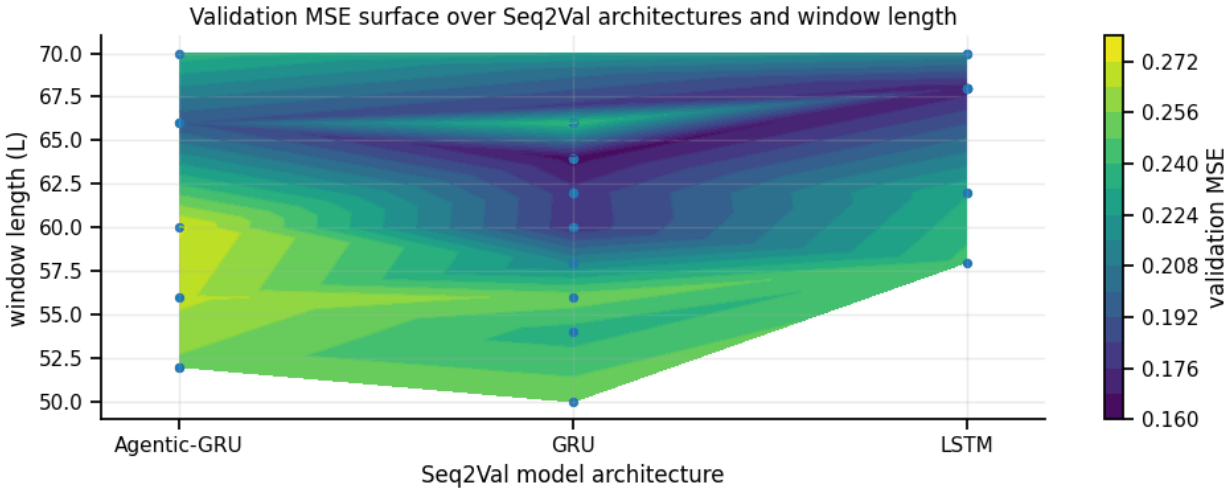


Figure 4.9: Validation loss surface for Seq2Val models as a function of window length (FD002).

Finally, Figure 4.10 presents the parity plot on the validation set for the best model, a GRU with window length 64 and stride 5. The predictions follow the diagonal closely, indicating good agreement between predicted and true RUL values, with larger deviations primarily occurring in the earlier life stages.

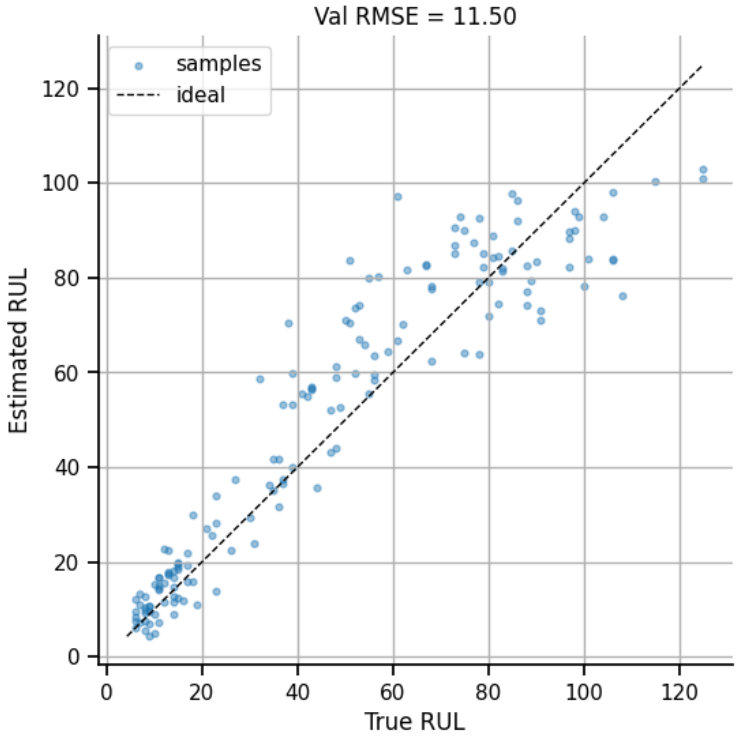


Figure 4.10: Parity plot of predicted versus true RUL on the validation set (FD002).

4.2 FEMTO Example

The FEMTO bearing dataset presents several challenges for data-driven RUL estimation. The data is strongly imbalanced, with most samples representing healthy operation, which motivates the use of resampling strategies. Measurements are acquired intermittently as short measurement intervals of 0.1 s, recorded approximately every 10 s. This sampling scheme, together with the high sampling frequency of the vibration signals (25.6 kHz), complicates temporal modeling by producing high-dimensional observations that are sparsely distributed in time. In addition, the vibration signals are affected by noise from multiple sources. Only a limited number of run-to-failure trajectories is available (12 for training and 5 for testing in this work), which makes it difficult to learn robust models that generalize across different operating conditions. This challenge is further exacerbated by the fact that bearing degradation is a complex, highly nonlinear process influenced by structural, operational, and environmental factors (Dhungana, Rykkje, and Lundervold 2025).

The FEMTO implementation example described below corresponds to the architecture selected in the hybrid execution mode, namely the best-performing configuration (Table D.1) obtained after agent-based initialisation followed by automated hyperparameter optimization. The resulting pipeline employs a dilated residual convolutional autoencoder for feature extraction and an eXtreme Gradient Boosting (XGBoost) regressor for remaining useful life prediction (Chen and Guestrin 2016).

4.2.1 Agentic Reasoning Process

For the FEMTO dataset, the agentic system was applied using dataset-specific user constraints to generate a suitable modeling pipeline. An example of the user-provided instruction for FEMTO is shown in Listing C.3.

4.2.2 Preprocessing

Due to the high sampling frequency and stochastic noise present in the FEMTO vibration signals, a lightweight denoising step is applied in the preprocessing step. Time synchronous moving average (Equation 3.2) with a small averaging window ($m = 2$) is used to suppress noise while preserving degradation-relevant signal characteristics.

To account for varying operating conditions, all sensor signals are normalised separately for each operating setting (Table 3.2). To reduce the dominance of long healthy operating periods, RUL values are clipped to a maximum of 200 cycles (RUL ≤ 200). This focuses the learning process on the region where degradation-related dynamics become observable. The remaining strong imbalance between healthy and degrading samples is addressed through aggressive resampling, where only 0.1% of healthy samples are retained. No explicit feature selection is applied, as representation learning is performed directly on the windowed time-series data.

4.2.3 Feature Engineering

Dilated convolutions extend standard one-dimensional convolutions by introducing a dilation factor r, which controls the spacing between kernel elements. Following Yu and Koltun (2016), a one-dimensional dilated convolution with dilation rate r is defined as

$$(x *_r w)(t) = \sum_{i=0}^{k-1} w(i)\, x(t - ri), \tag{4.1}$$

where $r = 1$ corresponds to standard convolution. Increasing r enlarges the receptive field without increasing the number of parameters or reducing temporal resolution.
To stabilise training of deeper dilated stacks, residual connections are used within the dilated convolutional blocks. These skip connections facilitate gradient propagation and allow the network to refine representations without discarding local signal information. The combination of dilated convolutions and residual connections therefore enables a large effective receptive field while maintaining training stability.
The autoencoder is trained using a reconstruction objective, minimising the MSE between the input signal x and its reconstruction $\hat{x}$,

$$\mathcal{L}_{\text{rec}} = \frac{1}{T} \sum_{t=1}^{T} \|x_t - \hat{x}_t\|^2 \tag{4.2}$$

where T denotes the number of temporal samples within a vibration window. This loss encourages the latent representation to preserve information relevant for signal reconstruction.

Application in the proposed Autoencoder

In this work, dilated residual convolutions are embedded within a one-dimensional convolutional autoencoder to extract degradation-related features from high-frequency vibration signals. After an initial strided convolutional encoder reduces the temporal resolution of each 0.1 s vibration cycle, a stack of residual dilated convolutional blocks with increasing dilation rates ($r \in \{1, 2, 4, 8\}$) is applied. This design allows the network to capture both short-term local patterns and longer-term temporal dependencies within a single cycle without additional downsampling. The resulting feature maps are aggregated using global average pooling, yielding a fixed-length representation independent of the temporal dimension. Finally, a linear projection maps the pooled representation to a compact latent vector, which is used as learned features for downstream RUL regression.

Latent Dimension

For FEMTO, each input sample consists of two vibration channels (`vib_h` and `vib_v`) segmented into cycle-wise windows of length $T = 2560$ samples (0.1 s at 25.6 kHz). The dilated convolutional autoencoder compresses each window into a latent vector $z \in \mathbb{R}^{16}$, which serves as the learned feature representation for downstream RUL regression.
To quantify the relationship between individual latent dimensions and the target, the mutual information between each latent feature z_j and RUL is computed (Equation B.1). Figure 4.11 reports the resulting scores across all 16 latent dimensions.

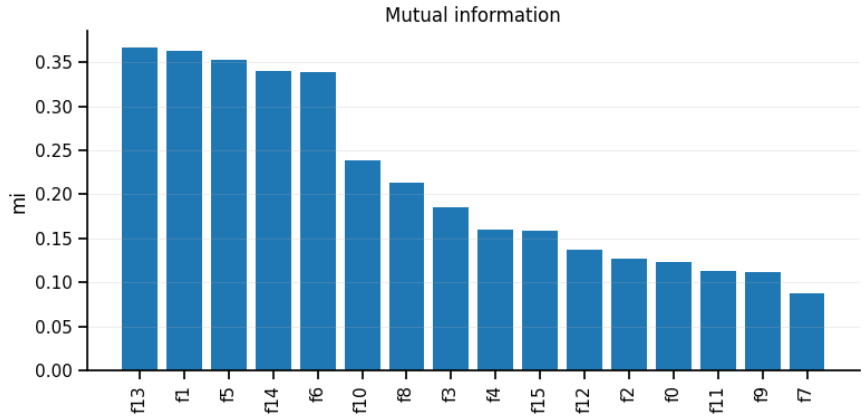


Figure 4.11: Mutual information between each latent feature and RUL for the FEMTO dataset.

The distribution of mutual information scores indicates that the learned representation is not uniformly informative: the most informative latent dimensions reach values above 0.3, while the least informative dimensions remain around 0.1. This suggests that a subset of latent features captures a stronger dependency with the degradation process, whereas other dimensions encode complementary variability that is less directly related to RUL. Overall, the presence of multiple high-scoring dimensions supports that the autoencoder learns degradation-relevant structure suitable for subsequent regression.

4.2.4 Regression

A single vibration cycle of 0.1 s provides only a limited temporal context and is therefore insufficient to capture the gradual degradation dynamics of rolling bearings. To incorporate longer-term temporal information, the *multi-window feature* strategy introduced in subsection 3.7.2 is applied to the learned latent representations.

Specifically, for each time step, the latent vectors extracted from the previous $M = 20$ consecutive cycles are concatenated, yielding a stacked feature representation of dimension $\mathbb{R}^{20\times16}$. This representation encodes short-term signal characteristics at the latent level while implicitly capturing degradation trends across multiple measurement intervals.

The resulting stacked features serve as input to an XGBoost regressor, which is well suited for modeling non-linear and heteroscedastic relationships. The convolutional autoencoder captures both spatial dependencies between sensor channels and short-term temporal dependencies within individual vibration cycles, while the multi-window stacking extends the temporal context to longer time horizons. Based on these representations, the regression model is able to predict the RUL effectively, as illustrated in Figure 4.12, achieving an RMSE of 21.5. Note that RUL values below 80 cannot be predicted, as the FEMTO test set does not contain samples with RUL less than 80 by design.

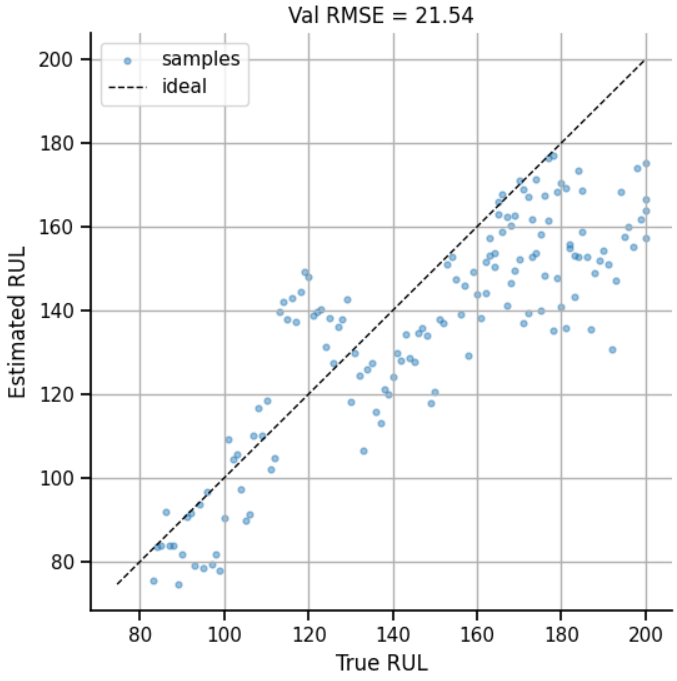


Figure 4.12: Predicted vs. true RUL for the FEMTO test set using the dilated residual convolutional autoencoder and an XGBoost regressor.

Chapter 5

Evaluation

This chapter reports, compares, and discusses the empirical results of the evaluated approaches.

5.1 Benchmark

This section evaluates the predictive performance of the proposed approaches on the C-MAPSS and FEMTO datasets. Results are reported in terms of RMSE (Equation 2.2) and summarised in Table 5.1. For the C-MAPSS datasets (FD001-FD004), the RUL target was clipped to a maximum of 125 cycles, while for FEMTO a maximum RUL of 200 was used. Hyperparameter optimization was performed with a fixed time budget of two hours and maximal 40 samples per run. All experiments were executed on a system equipped with four NVIDIA Tesla V100 PCIe GPUs, each providing 32 GB of memory. Agentic approaches accessed a LLM via the OpenAI application programming interface (API) (GPT-5) (OpenAI 2025).

5.1.1 Performance Comparison

As shown in Table 5.1, the hybrid agentic approach outperforms the benchmark methods on C-MAPSS FD002 and the FEMTO dataset. Across all experiments, the best-performing models, whether based on feature extraction followed by regression or on end-to-end Seq2Val architectures, correspond to predefined model classes rather than architectures newly generated by the Model Agent (A_m). For further details, see Table D.1.

Method	**FD001**	**FD002**	**FD003**	**FD004**	**FEMTO**
AutoCoevoRUL	15.61	17.79	15.62	16.34	33.07
AutoRUL	**9.21**	14.55	**8.14**	**12.58**	22.52
AutoML	<u>13.00</u>	13.48	<u>11.93</u>	<u>13.13</u>	31.43
Agentic (Standalone)	17.80	11.60	13.96	24.94	41.90
Agentic (Hybrid)	16.38	**<u>10.94</u>**	14.06	14.07	**<u>21.54</u>**

Table 5.1: Comparison of RMSE performance on C-MAPSS and FEMTO datasets. Including AutoCoevoRUL (T. Tornede et al. 2021) and AutoRUL (Zöller et al. 2023). Best overall results are shown in **bold**; best results among the proposed approaches are <u>underlined</u>.

Nevertheless, the architectures generated by the agentic approach achieve performance close to the selected baselines, particularly on C-MAPSS FD002 in the standalone agentic setting. Overall, the agentic approach proves especially effective in configuring the machine-learning pipeline, as reflected by the strong results of the hybrid variant. Finally, HPO leads to a significant performance improvement across most of the proposed approaches.

5.1.2 Selected Model Architectures

The best-performing model architectures for each dataset are summarised in Table 5.2. Across all C-MAPSS subsets, GRU-based Seq2Val models operating directly on raw sensor data are consistently selected, indicating that explicit feature extraction is not required for these datasets. In contrast, the FEMTO dataset benefits from careful feature extraction using a dilated CNN autoencoder with multi window enabled (subsection 3.7.2), followed by an XGBoost regressor.

Dataset	**Feature extractor**	**Regressor / Seq2Val**	**Windowing**	**Best method**
FD001	raw	GRU (Seq2Val)	$L = 54$	AutoML
FD002	raw	GRU (Seq2Val)	$L = 70$	Agentic (Hybrid)
FD003	raw	GRU (Seq2Val)	$L = 55$	AutoML
FD004	raw	GRU (Seq2Val)	$L = 96$	AutoML
FEMTO	Dilated CNN-AE	XGBoost	$L = 2560$, $M = 20$	Agentic (Hybrid)

Table 5.2: Best-performing model architectures per dataset, selected as the best result across AutoML, agentic standalone, and agentic hybrid approaches. L denotes the window length and M the number of consecutive windows.

5.2 Runtime and Cost Analysis

Table 5.3 summarises the runtime and inference cost comparison on the FEMTO dataset for the three evaluated approaches.

Method	**HPO Runtime**	**LLM Inference Cost [$]**	**Agentic Runtime**
AutoML	1 h 8 min	–	–
Agentic (Standalone)	one-shot	0.42	12 min 2 s
Agentic (Hybrid)	1 h 32 min	0.50	14 min 36 s

Table 5.3: Runtime and inference cost comparison on the FEMTO dataset.

The standalone agentic approach executes as a single-shot pipeline and does not rely on hyperparameter optimization, resulting in a shorter execution time compared to AutoML based methods. In contrast, the hybrid approach combines agentic configuration with HPO, leading to a higher total runtime.

The runtime introduced by agentic reasoning and execution remains moderate, on the order

of a few minutes. The reported inference costs correspond exclusively to LLM API usage during agent execution. A precise estimation of the total computational cost, including GPU utilization and energy consumption, is not provided.

Overall, the results indicate that agentic approaches introduce a limited runtime overhead while offering increased automation and flexibility. A comprehensive cost comparison is left for future work.

5.3 Error behavior and Additional Observations

This section analyzes additional error characteristics on the FEMTO dataset to complement the RMSE-based benchmark results. Specifically, the NASA score, mean absolute error (MAE), and bias are considered to provide insight into asymmetric error penalties, absolute prediction deviations, and systematic over- or underestimation behavior. Formal definitions of these metrics are provided in Appendix section D.2.

Method	**NASA Score**	**MAE**	**Bias**
AutoML	9257.8	25.9	−18.6
Agentic (Standalone)	8276.3	35.4	−26.8
Agentic (Hybrid)	866.3	17.6	−13.3

Table 5.4: Additional error metrics on the FEMTO dataset.

The results are consistent with the RMSE-based benchmark and confirm the superior performance of the hybrid agentic approach across all considered error metrics. In addition to achieving the lowest MAE, the hybrid approach exhibits the smallest absolute bias, indicating more balanced predictions and reduced systematic underestimation of the remaining useful life.

The substantially lower NASA score further suggests improved handling of late-life prediction errors, which are penalised more strongly in the PHM challenge metric. This indicates that the hybrid approach achieves not only higher average accuracy, but also more favourable error characteristics from a safety-oriented perspective.

In contrast, the standalone agentic approach shows a lower NASA score than the AutoML baseline but exhibits a higher MAE and a more pronounced negative bias. This suggests that its improved NASA score is primarily driven by conservative early predictions rather than improved overall accuracy.

5.3.1 Effect of Denoising on the FEMTO Dataset

Given the high noise level of the FEMTO vibration signals, the impact of denoising on prediction performance was evaluated. Applying temporal smoothing via TSMA resulted in a substantial reduction in RMSE for the hybrid agentic approach (dilated CNN-AE with XGBoost), decreasing

the RMSE from 28.96 to 23.04.

However, configurations using stronger smoothing ($m > 2$) exhibited degraded performance, indicating that denoising does not uniformly improve prediction accuracy. Excessive smoothing is likely to suppress informative high-frequency components that are relevant for bearing degradation modeling. These results highlight the sensitivity of model performance to preprocessing choices when applied to noisy bearing datasets.

5.4 Discussion

For the FEMTO dataset, several configurations converged to an almost constant prediction. Under a mean squared error objective, this behavior corresponds to predicting the unconditional expectation of the target, i.e. $\hat{y}(x) \approx \mathbb{E}[y]$. This situation arises when the conditional expectation $\mathbb{E}[y \mid x]$ cannot be reliably learned from the available input features (Bishop 2006, pp. 46–47). The multi-window approach (subsection 3.7.2) mitigated this issue by providing a larger temporal context, enabling the model to capture degradation-related dependencies that are not observable within single windows.

5.4.1 Robustness

The standalone AutoML baseline exhibited a failure rate of 5.26 %, meaning that the ML pipeline was not able to create a regression model for remaining useful life estimation in these cases. In comparison, the standalone agentic approach failed in 20.00 % of runs, corresponding to an absolute increase of 14.74 percentage points. The hybrid agentic approach showed a failure rate of 9.52 %, which corresponds to an absolute increase of 4.26 percentage points compared to the AutoML baseline. The lower failure rate of the hybrid approach compared to the standalone agentic setup can be explained by the presence of hyperparameter optimization. In the hybrid setting, a failed agent-generated model does not necessarily lead to a complete run failure, as the optimization process can still explore alternative configurations within the search space and recover from suboptimal or invalid initial proposals. The observed failure modes include:

- Dimension mismatches between generated model architectures and input data.
- Violation of user-specified constraints, such as unintended hyperparameter optimization.
- Invalid or inconsistent pipeline configurations, for example assigning a hybrid mode as a pipeline type.

5.4.2 Context Sensitivity of Agentic Approaches

Beyond quantitative performance differences, the agentic approaches were sensitive to the provided contextual information, including dataset descriptions, prior experimental results, and user instructions. This sensitivity influenced pipeline structure, preprocessing choices, and model selection, contributing to performance variability across runs.

Such context dependence is a known characteristic of LLM-based systems and poses challenges for reproducibility and robust evaluation (A. Tornede et al. 2024). Small changes in prompt formulation, available context, or instruction ordering can lead to substantially different modeling decisions, even when operating on the same dataset. Consequently, prompt and context engineering emerge as critical factors determining the effectiveness and stability of agentic approaches.

At the same time, this context sensitivity can be beneficial when domain knowledge is explicitly provided. In the FEMTO experiments, meaningful feature extraction required contextual guidance that specified appropriate inductive biases, such as the use of dilated CNN autoencoder for high-frequency vibration signals with large window lengths and the application of multi window representations (subsection 3.7.2) to capture temporal dependencies. When such information was absent, the agentic system frequently generated suboptimal pipelines, particularly in the feature extraction stage.

While the agent-based system exposes intermediate decisions, prompts, and generated artefacts, the internal reasoning process of the underlying LLM remains opaque. Only observable inputs, outputs, and optionally generated rationale summaries can be inspected. As a result, agentic decision-making is interpretable at the level of execution traces and produced configurations, but not at the level of the model's internal chain-of-thought.

In addition, the use of public benchmark datasets introduces the general possibility that such data may have been encountered during LLM pretraining, which complicates the attribution of performance gains. This further underscores the need for cautious interpretation of results and for evaluation protocols that explicitly account for contextual dependencies when assessing agentic systems.

5.4.3 Generalizability

The proposed RUL system achieved competitive performance on two substantially different benchmark datasets, indicating its ability to adapt across domains.

The FEMTO dataset is characterised by high-frequency vibration signals, a high sampling rate, low-dimensional sensor inputs, a small number of run-to-failure trajectories and a comparatively high noise level. In contrast, the C-MAPSS dataset contains lower-frequency measurements with higher-dimensional sensor data and more stable signal characteristics. Despite these fundamental differences, the same modeling framework was successfully applied to both datasets.

Achieving strong performance required dataset-specific modeling choices, including different preprocessing strategies, windowing schemes, and regression model classes. Across datasets with distinct signal characteristics, no single pipeline configuration performed optimally in all cases, highlighting the inherently dataset-dependent nature of RUL modeling.

Taken together, these results suggest that the proposed approach generalises at the system level by flexibly adapting its modeling pipeline to the data at hand, rather than relying on a fixed or dataset-specific configuration.

5.4.4 Summary

Overall, the results indicate that agentic guidance can be beneficial in certain modeling regimes, particularly for datasets such as FEMTO that are characterised by high noise levels, limited run-to-failure data, and complex temporal dependencies. While no causal analysis of the optimization dynamics was performed, the observed performance differences suggest that incorporating agent-generated structure can help guide the search process toward more suitable configurations.

One plausible explanation is that agentic guidance introduces implicit priors through model class selection, preprocessing decisions, and search space construction. This effect appears most pronounced in the hybrid setting, where agent-generated structure is combined with subsequent hyperparameter optimization. In contrast, standalone agentic approaches lack corrective optimization mechanisms, while pure AutoML relies on predefined, data-agnostic search spaces that may require expert tuning for strongly dataset-dependent scenarios.

At the same time, agentic approaches remain sensitive to the provided context and user instructions, which can lead to substantial variability in performance.

Chapter 6

Conclusion and Outlook

This thesis presented a unified framework for automated modeling in predictive maintenance with a focus on remaining useful life prediction. Three modeling paradigms were designed, implemented, and systematically evaluated: a classical AutoML baseline, a standalone agentic AI approach, and a hybrid agentic AutoML approach.

A common experimental framework was developed to ensure a fair and reproducible comparison across methods and datasets. All approaches were benchmarked on the C-MAPSS and FEMTO datasets, covering both simulated multivariate sensor data and real-world high-frequency vibration measurements.

The empirical evaluation provided a quantitative comparison based on the root mean square error and highlighted the strengths and limitations of each paradigm. In particular, the results demonstrate that agentic guidance can effectively complement structured AutoML pipelines, while fully standalone agent-based modeling remains less robust for reliable remaining useful life prediction under the investigated conditions.

Based on these findings, the hybrid agentic AutoML approach emerges as the most promising direction. It can lower the entry barrier for small and medium-sized enterprises adopting PdM systems, make RUL prediction more accessible to non-expert users, and enable the integration of domain knowledge into automated modeling workflows. Furthermore, the use of LLM-based pipeline and search-space generation allows greater flexibility when adapting to unseen datasets.

6.1 Key Findings

6.1.1 What Worked Well

The proposed ML pipeline (subsection 3.2.1) successfully enabled accurate prediction of the RUL across all evaluated datasets. Its modular design proved to be robust and generalizable, allowing the same software framework to be applied to datasets with substantially different characteristics and signal properties.

The standalone AutoML approach provided a strong and reliable baseline. It exhibited stable optimization behavior, consistent convergence, and a low failure rate across multiple runs and datasets, demonstrating the effectiveness of structured search spaces for automated RUL modeling.

The agentic approach demonstrated several strengths beyond classical AutoML configurations. Most notably, it enabled dataset-aware reasoning by explicitly analysing dataset characteristics and by incorporating user-provided information and high-level instructions. This allowed the system to align the generated modeling pipelines more closely with the intended experimental constraints, reducing manual configuration effort and improving transparency compared to fully implicit AutoML workflows.

Importantly, this behavior moves automated PdM systems towards a usage paradigm in which users with limited data science expertise can still construct meaningful and competitive RUL models. High-level guidance can be expressed in natural language, while the agentic system translates these requirements into concrete pipeline configurations and modeling decisions.

Acting as a virtual data scientist, the underlying LLM contributed prior knowledge about established RUL modeling practices. This included proposing promising search spaces, suggesting suitable preprocessing strategies, and generating novel model architectures that were not part of the predefined AutoML search space. As a result, agentic guidance complemented structured optimization by steering the search toward more promising configurations.

From a development-effort perspective, agentic approaches reduced the manual effort required for pipeline configuration and search-space design. At the same time, they increased debugging and validation effort due to the stochastic nature of LLM outputs, particularly when constraint violations or invalid configurations occurred.

6.1.2 What Did Not Work Well

Despite its strong baseline performance, the AutoML approach exhibited structural limitations. The use of a fixed and predefined search space restricts data-dependent modeling decisions and limits adaptability to dataset-specific characteristics. Conversely, manually designing or refining AutoML configurations requires substantial data science expertise, which contradicts the goal of fully automated and broadly accessible predictive maintenance systems.

Agentic approaches showed an increased failure rate compared to the AutoML baseline, particularly in the standalone agentic setting, as discussed in subsection 5.4.1.

In addition, the underlying LLM occasionally violated user-specified constraints or produced invalid configurations. These behaviors highlight current limitations of autonomous LLM-based code and pipeline generation, particularly with respect to reliability, constraint adherence, and

reproducibility.

6.1.3 Why These Results Matter

The results of this work indicate that, within the evaluated experimental setting, agentic AI does not act as a direct replacement for AutoML, but rather as a complementary approach. While AutoML provides robust optimization within well-defined search spaces, agentic guidance adds flexibility by enabling dataset-aware reasoning and the incorporation of high-level user intent.

At the same time, the findings highlight the limitations of both baseline AutoML configurations and unconstrained agentic AI. Fixed search spaces in AutoML restrict adaptability to dataset-specific characteristics, whereas blind agentic approaches suffer from increased failure rates and reduced robustness when operating without structured optimization mechanisms.

Crucially, the evaluation reveals the central role of prompt and context engineering in steering agentic approaches towards effective modeling decisions. Agent performance depends strongly on the availability of explicit user guidance that encodes domain-relevant assumptions about the data and the modeling objective. Without such guidance, the agentic system struggles to infer appropriate pipeline construction solely from data statistics.

These findings suggest that agentic AI systems for automated PdM should be understood as human-in-the-loop tools. Effective deployment requires carefully designed prompt and context interfaces that translate domain knowledge into actionable constraints.

6.2 Critical Discussion

This thesis does not aim to exhaustively explore all agentic architectures; instead, it compares representative approaches and outlines directions for future research.

6.2.1 Methodological Limitations

A fundamental limitation of the AutoML-based approach lies in the use of a fixed and predefined search space. As noted by Zöller et al. (2023), such constraints prevent systems like AUTORUL from becoming universal tools, since it is always possible for domain specialists to construct datasets whose characteristics fall outside the representational capacity of the search space.

Furthermore, the statistical robustness and generalizability of the evaluated RUL models are limited by the size of the available datasets.

Despite the increased flexibility introduced by agentic components, the overall system still operates within predefined modeling pipeline, such as window-based learning and supervised regression and they may not be optimal for all types of degradation processes scenarios.

6.2.2 Agentic AI Specific Risks

A potential risk of the agentic approaches arises from the use of large language models that may have been exposed to public benchmark datasets, such as FEMTO, during pretraining.

In addition, agentic reasoning remains largely opaque. While prompts, generated configurations, and execution traces can be inspected, the internal decision-making process of the underlying LLM is not directly observable, limiting interpretability and complicating systematic debugging.
Agentic approaches further exhibit an increased susceptibility to pipeline failures. Because modeling decisions, architecture generation, and configuration synthesis are delegated to an LLM, invalid or inconsistent outputs, such as dimension mismatches, violated constraints, or incompatible pipeline specifications, can lead to failed runs. As discussed in subsection 5.4.1, this brittleness is reflected in noticeably higher empirical failure rates for agentic configurations compared to the classical AutoML baseline.

Agentic approaches also incur increased computational and operational costs. The combination of LLMs with automated optimization introduces additional inference overhead, and longer end-to-end runtimes compared to classical AutoML pipelines.

Finally, the effectiveness of agentic systems depends strongly on the availability of high-quality user guidance. Without sufficiently detailed domain knowledge encoded in prompts and context, agent-generated pipelines may remain suboptimal, limiting the degree of fully autonomous operation.

6.2.3 AutoML vs. LLM-Based Generation

In terms of predictive accuracy and robustness, the HPO-based AutoML baseline mostly outperformed the standalone agentic approach (Table 5.1). Structured optimization within a predefined search space yielded stable convergence behavior and lower failure rates across datasets.

Notably, the best-performing model on the FEMTO dataset was obtained using the hybrid approach, in which agentic guidance was combined with structured HPO.

At the same time, LLM-based pipeline, search-space, and model generation substantially increased modeling flexibility. By incorporating domain knowledge, agentic approaches lowered the barrier for non-expert users and enabled the exploration of modeling choices that would otherwise require significant data science expertise.

However, this flexibility comes at the cost of reduced reproducibility and increased debugging complexity. Agentic approaches are inherently less reproducible due to the stochastic nature of LLM outputs, and failures are more difficult to diagnose because of limited transparency in the model's internal reasoning. As a result, careful evaluation, validation, and testing of generated code and configurations are required to ensure correctness, increasing overall software complexity and maintenance effort.

6.3 Practical Recommendations

Based on the empirical findings of this work, hybrid agentic AutoML pipelines are recommended for automated PdM tasks. The results indicate that neither classical AutoML nor standalone agentic approaches alone provide an optimal balance between robustness, flexibility, and automation.

Agentic AI is particularly well suited for analysing dataset characteristics and user constraints, and for translating these into informed modeling choices such as search-space design, preprocessing strategies, model class selection, feature engineering, and the proposal of new regression architectures beyond predefined ones. In this role, the agent acts as a virtual domain-aware assistant, reducing manual configuration effort and lowering the entry barrier for non-expert users.

AutoML, in contrast, should be used for robust optimization within these agent-defined structures. Once the search space and pipeline structure are specified, AutoML provides reliable and reproducible hyperparameter optimization, stable convergence behavior, and effective resource utilization through multi-fidelity optimization strategies. This separation of responsibilities combines the flexibility of agentic reasoning with the stability and robustness of structured optimization.

For deployment in practical PdM settings, it is further recommended to use highly configurable but predefined ML pipeline structures. Such designs improve reproducibility, simplify debugging, and reduce operational risk, while still allowing sufficient flexibility through agent-guided configuration.

Finally, it should be noted that this thesis does not cover all possible agentic design choices. Given the rapid development of agentic AI, additional opportunities exist beyond the scope of this work, particularly with respect to iterative decision loops, and case-based reasoning. These directions are discussed further in the outlook (section 6.4) and in relation to recent state-of-the-art agentic AutoML systems (subsection 2.4.3).

6.4 Outlook

A key limitation of the current agentic approach is that pipeline decisions regarding preprocessing, feature engineering, and model selection are made only once at the beginning of the workflow. This single-shot design may be insufficient for complex or noisy datasets, where the suitability of modeling choices can only be assessed after observing their effect on training behavior.

A promising direction for future work is the development of iterative agentic workflows in which the agent inspects intermediate artefacts and adapts its decisions accordingly. Such workflows could iteratively revise preprocessing, feature engineering, and model selection based on observed performance, thereby aligning agentic AutoML more closely with human expert workflows and improving robustness across heterogeneous datasets.

Iterative refinement may be realised either by combining agentic guidance with classical HPO, or

through fully agent-driven optimization loops such as AgentHPO (S. Liu, Gao, and Y. Li 2025). In addition, case-based reasoning, as described by Guo et al. (2024), offers a mechanism to reuse model templates and hyperparameters from prior tasks, reducing search cost and improving optimization stability. Further extensions include enabling model agents to generate dataset-specific feature extraction architectures.

Beyond these methodological extensions, the proposed system should be regarded as a research prototype rather than a fully production-ready solution. Future work should therefore focus on improving the reliability of LLM-based pipeline generation. While this thesis introduces syntactic and minimal structural validation (e.g. YAML configuration syntax checks and basic schema verification), additional semantic validation layers are required to ensure that generated preprocessing steps, model choices, and hyperparameter ranges are consistent with dataset characteristics and user constraints. Model-based evaluation strategies, such as *LLM-as-a-Judge*, may support this process.

Classical software testing techniques should also be integrated more systematically into agentic AutoML workflows. Unit tests can validate generated code components, while integration tests ensure correct interaction between pipeline stages. Together, these measures are essential to enable reliable and industrially applicable LLM-driven predictive maintenance systems.

Finally, future implementations should include a user interface in which a prompt agent actively guides users to provide the necessary contextual information, instead of relying on predefined prompt templates. Real-world deployments must also account for the frequent absence of complete run-to-failure data, which necessitates strategies capable of learning from partial degradation trajectories.

Abbreviations

AE autoencoder

AI artificial intelligence

API application programming interface

ASHA asynchronous successive halving algorithm

AutoML automated machine learning

AUTORUL Automated Machine Learning for Remaining Useful Life Predictions

BO Bayesian optimization

BOHB Bayesian optimization and Hyperband

CASH combined algorithm selection and hyperparameter optimization

CBR case-based reasoning

CNN convolutional neural network

EI expected improvement

EMA exponential moving average

GRU gated recurrent unit

HB Hyperband

HPO hyperparameter optimization

IoT Internet of Things

LLAMBO Large Language Models to enhance Bayesian Optimization

LLM large language model

LSTM long short-term memory

MA moving average

MAE mean absolute error

MBE mean bias error

MCTS Monte Carlo Tree Search

ML machine learning

MLP multilayer perceptron

MSE mean squared error

MTS multivariate time series

PdM predictive maintenance

PSD power spectral density

RMSE root mean square error

RUL remaining useful life

SaaS Software as a Service

SELA Tree-Search Enhanced LLM Agents

Seq2Val sequence-to-value

SH successive halving

SMBO sequential model-based optimization

SMEs small and medium-sized enterprises

SNR signal-to-noise ratio

TCN temporal convolutional network

TPE Tree-structured Parzen Estimator

TSA time domain synchronous average

TSMA time synchronous moving average

UTS univariate time series

XGBoost eXtreme Gradient Boosting

List of Figures

List of Tables

Listings

References

Akiba, Takuya et al. (July 25, 2019). *Optuna: A Next-generation Hyperparameter Optimization Framework*. DOI: 10.48550/arXiv.1907.10902.

Arias Chao, Manuel et al. (Jan. 13, 2021). "Aircraft Engine Run-to-Failure Dataset under Real Flight Conditions for Prognostics and Diagnostics". In: *Data*. DOI: 10.3390/data6010005.

Asif, Owais et al. (2022). "A Deep Learning Model for Remaining Useful Life Prediction of Aircraft Turbofan Engine on C-MAPSS Dataset". In: *IEEE Access*. DOI: 10.1109/ACCESS.2022.3203406.

Bergstra, James et al. (2011). "Algorithms for hyper-parameter optimization". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., pp. 2546–2554. ISBN: 9781618395993.

Bigler, Joshoua (2025a). *Distributed IoT Platform*. Project work, OST - Eastern Switzerland University of Applied Sciences. URL: https://raw.githubusercontent.com/joshoua-bigler/IoT_SaaS/v1.0.0/docs/distributed_iot_platform.pdf (visited on 12/17/2025).

— (2025b). *Machine Learning Techniques for Distributed IoT Platform*. Project work, OST - Eastern Switzerland University of Applied Sciences. URL: https://raw.githubusercontent.com/joshoua-bigler/IoT_SaaS/v1.0.0/docs/machine_learning_techniques_for_distributed_iot_platform.pdf (visited on 12/17/2025).

Bishop, Christopher M. (2006). *Pattern recognition and machine learning*. Information science and statistics. New York: Springer. ISBN: 978-0-387-31073-2.

Chen, Tianqi and Carlos Guestrin (Aug. 13, 2016). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794. DOI: 10.1145/2939672.2939785.

Chi, Yizhou et al. (Oct. 22, 2024). *SELA: Tree-Search Enhanced LLM Agents for Automated Machine Learning*. DOI: 10.48550/arXiv.2410.17238.

Christ, Maximilian et al. (Sept. 2018). "Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh - A Python package)". In: *Neurocomputing*. DOI: 10.1016/j.neucom.2018.03.067.

Darban, Zahra Zamanzadeh et al. (Jan. 31, 2025). "Deep Learning for Time Series Anomaly Detection: A Survey". In: *ACM Computing Surveys*. DOI: 10.1145/3691338.

Dhungana, Hariom, Thorstein Rykkje, and Alexander S. Lundervold (2025). "Bearing Prognostics Using the PRONOSTIA Data: A Comparative Study". In: *IEEE Access*. DOI: 10.1109/ACCESS.2025.3551772.

Guo, Siyuan et al. (May 28, 2024). *DS-Agent: Automated Data Science by Empowering Large Language Models with Case-Based Reasoning*. DOI: 10.48550/arXiv.2402.17453.

Gupta, Rajeev Kumar, Jay Nakum, and Punit Gupta (2025). "A Machine Learning Approach for Turbofan Jet Engine Predictive Maintenance". In: *Procedia Computer Science*. DOI: 10.1016/j.procs.2025.03.317.

Hanchuan Peng, Fuhui Long, and C. Ding (Aug. 2005). "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. DOI: 10.1109/TPAMI.2005.159.

Hollmann, Noah, Samuel Müller, and Frank Hutter (Sept. 28, 2023). *Large Language Models for Automated Data Science: Introducing CAAFE for Context-Aware Automated Feature Engineering*. DOI: 10.48550/arXiv.2305.03403.

Hutter, Frank, Lars Kotthoff, and Joaquin Vanschoren, eds. (2019). *Automated Machine Learning: Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Cham: Springer International Publishing. DOI: 10.1007/978-3-030-05318-5.

Jamieson, Kevin and Ameet Talwalkar (Feb. 27, 2015). *Non-stochastic Best Arm Identification and Hyperparameter Optimization*. DOI: 10.48550/arXiv.1502.07943.

Kapoor, Sayash et al. (July 1, 2024). *AI Agents That Matter*. DOI: 10.48550/arXiv.2407.01502.

Krishnan, Naveen (Mar. 16, 2025). *AI Agents: Evolution, Architecture, and Real-World Applications*. DOI: 10.48550/arXiv.2503.12687.

LangChain (2026a). *LangChain: Framework for building agents and LLM-powered applications*. Accessed: 2026-01-18. URL: https://github.com/langchain-ai/langchain.

— (2026b). *LangGraph: Graph-based orchestration for agents*. Accessed: 2026-01-18. URL: https://github.com/langchain-ai/langgraph.

Li, Liam et al. (Mar. 16, 2020). *A System for Massively Parallel Hyperparameter Tuning*. DOI: 10.48550/arXiv.1810.05934.

Li, Lisha et al. (June 18, 2018). *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. DOI: 10.48550/arXiv.1603.06560.

Liaw, Richard et al. (July 13, 2018). *Tune: A Research Platform for Distributed Model Selection and Training*. DOI: 10.48550/arXiv.1807.05118.

Liu, Siyi, Chen Gao, and Yong Li (Feb. 26, 2025). *Large Language Model Agent for Hyper-Parameter Optimization*. DOI: 10.48550/arXiv.2402.01881.

Liu, Tennison et al. (Mar. 8, 2024). *Large Language Models to Enhance Bayesian Optimization*. DOI: 10.48550/arXiv.2402.03921.

Nectoux, Patrick et al. (June 2012). "PRONOSTIA: An experimental platform for bearings accelerated degradation tests". In: Conference on Prognostics and Health Management. Pp. 1–8.

OpenAI (2025). *OpenAI API Documentation*. URL: https://platform.openai.com/docs (visited on 12/04/2025).

Patil, Avinash and Aryan Jadon (May 29, 2025). *Advancing Reasoning in Large Language Models: Promising Methods and Approaches*. DOI: 10.48550/arXiv.2502.03671.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Peel, Leto (Oct. 2008). "Data driven prognostics using a Kalman filter ensemble of neural network models". In: *2008 International Conference on Prognostics and Health Management*, pp. 1–6. DOI: 10.1109/PHM.2008.4711423.

PwC (2017). *Predictive Maintenance 4.0: Predict the unpredictable*. URL: https://www.pwc.be/en/documents/20171016-predictive-maintenance-4-0.pdf (visited on 12/14/2025).

Scharei, Kristina, Florian Heidecker, and Maarten Bieshaar (Jan. 15, 2020). *Knowledge Representations in Technical Systems – A Taxonomy*. DOI: 10.48550/arXiv.2001.04835.

Thornton, Chris et al. (Mar. 6, 2013). *Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms*. DOI: 10.48550/arXiv.1208.3719.

Tornede, Alexander et al. (Feb. 21, 2024). *AutoML in the Age of Large Language Models: Current Challenges, Future Opportunities and Risks*. DOI: 10.48550/arXiv.2306.08107.

Tornede, Tanja et al. (June 26, 2021). "Coevolution of remaining useful lifetime estimation pipelines for automated predictive maintenance". In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 368–376. DOI: 10.1145/3449639.3459395.

Trirat, Patara, Wonyong Jeong, and Sung Ju Hwang (June 6, 2025). *AutoML-Agent: A Multi-Agent LLM Framework for Full-Pipeline AutoML*. DOI: 10.48550/arXiv.2410.02958.

Virtanen, Pauli et al. (2020). *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*. DOI: 10.1038/s41592-019-0686-2.

Welch, P. (June 1967). "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms". In: *IEEE Transactions on Audio and Electroacoustics*, pp. 70–73. DOI: 10.1109/TAU.1967.1161901.

Yu, Fisher and Vladlen Koltun (Apr. 30, 2016). *Multi-Scale Context Aggregation by Dilated Convolutions*. DOI: 10.48550/arXiv.1511.07122.

Zhang, Lun and Niaoqing Hu (2019). "Time Domain Synchronous Moving Average and its Application to Gear Fault Detection". In: *IEEE Access*, pp. 93035–93048. DOI: 10.1109/ACCESS.2019.2927762.

Zöller, Marc-André et al. (Oct. 1, 2023). "Automated Machine Learning for Remaining Useful Life Predictions". In: *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 2907–2912. DOI: 10.1109/SMC53992.2023.10394031.

Appendix A

Datasets

A.1 C-MAPSS Turbofan Engine Dataset

Sensor	Parameter	Description with units	Trend
1	T2	Total Temperature in fan inlet (°R)	∼
2	T24	Total Temperature at LPC outlet (°R)	I
3	T30	Total Temperature at HPC outlet (°R)	I
4	T50	Total Temperature at LPT outlet (°R)	I
5	P2	Pressure at fan inlet (psia)	∼
6	P15	Total pressure in bypass-duct (psia)	∼
7	P30	Total pressure at HPC outlet (psia)	D
8	Nf	Physical fan speed (rpm)	I
9	Nc	Physical core speed (rpm)	I
10	Epr	Engine pressure ratio (—)	∼
11	Ps30	Static pressure at HPC outlet (psia)	I
12	Phi	Ratio of fuel flow to Ps30 (psi)	D
13	NRf	Corrected fan speed (rpm)	I
14	Nrc	Corrected core speed (rpm)	I
15	BPR	Bypass ratio (—)	I
16	farB	Burner fuel air ratio (—)	I
17	htBleed	Bleed enthalpy (—)	I
18	NF-dmd	Demanded fan speed (rpm)	∼
19	PCNR-dmd	Demanded corrected fan speed (rpm)	∼
20	W31	HPT coolant bleed (lbm/s)	D
21	W32	LPT coolant bleed (lbm/s)	D

Table A.1: Sensor channels of the C/MAPSS dataset. The trend symbols follow (Asif et al. 2022): "∼" indicates irregular behavior, "I" denotes an increasing pattern over time, and "D" marks a decreasing pattern.

A.2 FEMTO Bearing Dataset

Attribute	Value
Number of rows	19,287,040
Number of units	6
Unit column	`unit`
Time column	`cycle`
Sensor columns	`vib_h`, `vib_v`
Operating setting column	`os`
Numeric columns	`cycle`, `vib_h`, `vib_v`, `os`
Categorical columns	`unit`
Sampling rate (Hz)	25600

Table A.2: Automatically inferred schema for the FEMTO dataset.

Appendix B

Methodology Details

B.1 ML Pipeline

```
dataset:
  name: '${dataset.name}'
  version: '${dataset.version}'

pipeline:
  type: { space: choice, values: '${available_pipelines_from_source}' }

criterion:
  kind: ${criterion_kind}
  lam: ${criterion_lam}
  reduction: ${criterion_reduction}

preprocessing:
  cleanup:
    drop: ${drop_columns}
  imputation:
    method: ${imputation_method}
  denoising:
    method: ${denoising_method}
    ignore: ${denoising_ignore}
    group_by: ${denoising_group_by}
    params: ${denoising_params}
  normalization:
    method: ${normalization_method}
    norm_type: ${normalization_type}
    os_cols: ${normalization_os_cols}
    group_by: ${normalization_group_by}
    ignore: ${normalization_ignore}
    sensor_cols: ${normalization_sensor_cols}
    kmax: ${normalization_kmax}
    min_level_frac: ${normalization_min_level_frac}
    add_mode_onehot: ${normalization_add_mode_onehot}

  select_by_correlation:
    apply: ${select_by_correlation_apply}
    method: ${select_by_correlation_method}
    correlation_threshold: ${select_by_correlation_threshold}
    ignore: ${select_by_correlation_ignore}
    mi:
      n_neighbors: ${select_by_correlation_mi_n_neighbors}
```

```
      random_state: ${select_by_correlation_mi_random_seed}
      normalize: ${select_by_correlation_mi_normalize}

  select_topk_mi:
    apply: ${select_topk_mi_apply}
    target_col: ${select_topk_mi_target_col}
    top_k: ${select_topk_mi_top_k}
    ignore: ${select_topk_mi_ignore}
    n_neighbors: ${select_topk_mi_n_neighbors}
    random_state: ${select_topk_mi_random_state}

  target_scaling:
    method: ${target_scaling_method}
    column: ${target_scaling_column}
    clip: ${target_scaling_clip}

  resampling:
    apply: ${resample_apply}
    group_by: ${resample_group_by}
    keep_healthy_fraction: ${resample_keep_healthy_fraction}
    clip: ${target_scaling_clip}
    rul_col: ${resample_rul_col}

feature_engineering:
  window:
    drop_last: false
    drop_first: false
    axis: 0
    group_by: unit
    size: { space: qrandint, low: '${auto_from_seq_len_min}', high: '${auto_from_seq_len_max}', q: '${auto_from_seq_len_q}' }
    stride: { space: qrandint, low: '${auto_from_seq_len_stride_low}', high: '${auto_from_seq_len_stride_high}', q: '${auto_from_seq_len_stride_q}' }

feature_extraction:
  model: { space: choice, values: '${fe_models_from_source}' }
  models: '{}'

regression:
  model: { space: choice, values: '${reg_models_from_source}' }
  models: '{}'

seq2val:
  model: { space: choice, values: '${seq2val_models_from_source}' }
  trainer: '{}'
  models: '{}'

paths:
  root: ./data
  artifacts: ./artifacts
```

Listing B.1: ML pipeline configuration template

B.2 Metadata

```
{
  "columns": {
```

```
    "target_col": "rul",
    "unit_col": "unit"
  },
  "impute_method": "none",
  "normalization": {
    "method": "unique"
  },
  "resampling_params": {
    "clip": 125,
    "train_keep_healthy_fraction": 0.4,
    "test_keep_healthy_fraction": 0.4,
    "seed": 42
  },
  "sensor_selection": {
    "correlation": {
      "method": "pearson",
      "corr_threshold": 0.9,
      "dropped_corr": ["s14"],
      "ignored": ["opc_mode", "os1", "os2", "os3", "unit"],
      "kept": ["s2", "s3", "s4", "..."]
    },
    "mutual_information": {
      "top_k": 13,
      "dropped": ["s6"],
      "kept": ["s11", "..."]
    }
  },
  "y_scaler": {
    "method": "standard",
    "params": {
      "mu": 75.21,
      "sd": 41.07
    }
  }
}
```


Listing B.2: Simplified example of pipeline metadata produced after preprocessing (C-MAPSS FD001).

B.2.1 Feature Selection

Mutual Information

For discrete variables X and Y with joint distribution $p(x, y)$ and marginals $p(x)$ and $p(y)$, the mutual information is

$$MI(X;Y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y) \log \left(\frac{p(x,y)}{p(x)\, p(y)} \right) \tag{B.1}$$

Pearson Correlation

For a sample $\{(x_t, y_t)\}_{t=1}^{n}$ with means $\bar{x}$ and $\bar{y}$, the empirical Pearson correlation is

$$\hat{r}_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{B.2}$$

Appendix C

Agentic AI System Details

C.1 Agentic AI Context

C.1.1 Prompt Agent

```
# Prompt Agent

## Role
Collect dataset context, user feedback, and past optimization results.
Produce structured prompts for the Model and Config Agents.

## Responsibilities
- Load and summarise feedback files.
- Inspect Ray Tune results.
- Derive diagnostics (noise, collapse, bias).
- Output prompts for A_m and A_c.
- Provide hints: best scores, trends, parity insights.

## Output
{
  "config_prompt": {...},
  "model_prompt": {...},
  "hints": {...},
  "phase": "need_model" | "need_config"
}
```

Listing C.1: Simplified system prompt for the Prompt Agent (A_{p}).

```
# CMAPPS
The CMAPPS dataset (Commercial Modular Aero-Propulsion System Simulation) is a
widely used benchmark dataset for prognostics and health management (PHM)
research, particularly in the context of aircraft engine degradation and
remaining useful life (RUL) prediction. It was developed by NASA and is
designed to simulate the behavior of a commercial aircraft engine under
various operating conditions and fault scenarios.

## Preprocessing Recommendations
- Keep healthy fraction around 0.4
- normalization with per operational setting using opc_mode (kmax=6)
- rul clip at 125
- apply mutual information based feature selection
- apply select by correlation
- apply target scaling
```

```
- Prefer seq2val architecture e.g. lstm, gru etc.
- Prefer deep learning based models
- Don't use tsfresh or flatten based feature extraction
- Max window size around 50-70
- Max stride around 5-10
```

Listing C.2: Instruction input provided to the Prompt Agent (A_{p}); C-MAPSS FD002.

```
# FEMTO Feature Extraction (CNN / Dilated CNN AE) + Regression Prompt

## Dataset Context
- FEMTO / PRONOSTIA bearing dataset
- Natural degradation with very limited run-to-failure samples
- 17 runs total (11 train, 6 test)
- 3 operating settings (400-500 N, 1200-1800 rpm)
- Signals: vib_h, vib_v sampled at 25.6 kHz
- One cycle = 2560 samples (0.1 s), cycle interval = 10 s

## Hard Constraints (must not be violated)
- Feature extraction method: CNN-based autoencoder
  - standard CNN AE or dilated CNN AE
- Sensor columns: vib_h, vib_v
- Target column: rul
- RUL clipping: 200
- Resampling:
  - group_by: unit
  - keep_healthy_fraction ~ 0.001
- Normalization:
  - normalize strictly per operational setting (per-os)
- Windowing:
  - by_cycle = true
  - fast = true
  - no window size or stride configuration
  - encode_os = false
- Use TSMA denoising with m = 2
- Use multi-windowing:
  - stack multiple latent windows before regression (search space between 10 and 30)
  - stack multipe windows for seq2val models (search space between 10 and 30)
- target_scaling: minmax or standard (target only)
- Drop os after preprocessing

## Key Challenges
- Extremely few degradation trajectories
- Strong operating-setting dependency
- High stochasticity in degradation patterns
- Tsfresh with XGBoost tends to create constant predictions

## Feature Extraction Strategy
- Dilated convolutions are allowed to increase receptive field
- Train AE only to reconstruction stability (not overfitting)

## Regression Strategy
- Input: stacked latent vectors from multi-windowing
- Preferred regressors:
  1) TCN (seq2val)
  2) XGBoost
  3) Small MLP

## Hints
- Seq2Val TCN with multi window m = 13 is a strong baseline
```

```
- The window length is large (2560 samples) * m (multi window)
- Prefer depth over width (small channel counts, more layers if needed).

## Objective
- Preserve degradation variance across operating settings
- Avoid regression-to-mean behavior
- Prefer early predictions over late predictions
```

Listing C.3: Instruction input provided to the Prompt Agent (A_{p}); FEMTO.

C.1.2 Model Agent

```
# Model Agent

## Role
Generate one or more candidate models suitable for the dataset.

## Responsibilities
- Select or create model architectures.
- Support tabular and sequence models.
- Produce deterministic Python modules.
- Register models using the system adapters.

## Output
{
  "model_code": "<python module>",
  "metadata": {...}
}
```

Listing C.4: Simplified system prompt for the Model Agent (A_{m}).

```
# Prompt Agent Context
task: seq2val_rul
preferred_model: gru

inputs:
  features: [s1, s2, ..., s21]
  aux: [os1, os2, os3]

architecture:
  type: gru
  hidden_size: 128
  num_layers: 2
  dropout: 0.2
  bidirectional: false

training:
  loss: smooth_l1
  optimizer:
    type: adam
    lr: 1e-3
    weight_decay: 1e-5
  batch_size: 128
  epochs: 60

windowing:
  size: 64
  stride: 8
```

```
  target: last

constraints:
  rul_clip: 125
  normalization: per_op_setting_opc_mode (kmax=6)
  avoid: [tsfresh, flatten]
  feature_selection: [mutual_information, correlation]

# Prompt Agent Hints
key_trends:
  - Seq2Val models (LSTM/GRU) perform best on C-MAPSS
  - Normalize per operating condition using OPC mode clustering (kmax = 6)
  - Apply RUL clipping (125) and retain a healthy fraction of ~0.4
  - Typical window size: 50-70, stride: 5-10
  - Prefer MI and correlation-based feature selection; avoid tsfresh/flatten
```

Listing C.5: Example guidance produced by the Prompt Agent (A_{p}) for the Model Agent (A_{m}); C-MAPSS FD002.

```
class GRUSeq2ValRUL(nn.Module):
  ''' GRU-based Seq2Val model for RUL prediction on CMAPSS-like data.

      Input shape: [batch, time, features]
      Output shape: [batch, 1] (or [batch, Q] if quantiles provided)
  '''

  def __init__(self,
               in_feats: int,
               hidden_size: int = 128,
               num_layers: int = 2,
               dropout: float = 0.2,
               bidirectional: bool = False,
               quantiles: Optional[List[float]] = None,
               **kwargs: Any) -> None:
    super().__init__()
    seed = int(kwargs.get('seed', 0))
    torch.manual_seed(seed)
    self.input_dim = int(in_feats)
    self.hidden_size = int(hidden_size)
    self.num_layers = int(num_layers)
    self.dropout = float(dropout)
    self.bidirectional = bool(bidirectional)
    self.quantiles = quantiles
    self.rnn = nn.GRU(input_size=self.input_dim,
                      hidden_size=self.hidden_size,
                      num_layers=self.num_layers,
                      dropout=self.dropout if self.num_layers > 1 else 0.0,
                      bidirectional=self.bidirectional,
                      batch_first=True)
    out_dim = len(self.quantiles) if (self.quantiles is not None and len(self.quantiles) >
    0) else 1
    last_feat_dim = self.hidden_size * (2 if self.bidirectional else 1)
    self.head = nn.Sequential(nn.Dropout(self.dropout), nn.Linear(last_feat_dim, out_dim))

  def forward(self, x: torch.Tensor) -> torch.Tensor:
    ''' Forward pass.

        Parameters
        ----------
```

```
        x : torch.Tensor
          Tensor of shape [B, T, F].
    '''
    assert x.dim() == 3, f'Expected 3D input [B,T,F], got {tuple(x.shape)}'
    assert x.size(-1) == self.input_dim, (
        f'Input feature dim mismatch: expected {self.input_dim}, got {x.size(-1)}')
    out, _ = self.rnn(x)
    last = out[:, -1, :]
    y = self.head(last)
    return y
```

Listing C.6: Model Agent (A_{m}) generated Seq2Val model for C-MAPSS FD002.

C.1.3 Config Agent

```
# Config Agent

## Role
Construct the full YAML configuration for the RUL pipeline.

## Responsibilities
- Integrate dataset schema and model report.
- Fill preprocessing, feature, model, and search-space fields.
- Apply dataset-specific constraints (RUL clipping, OS grouping).
- Validate completeness of the configuration.

## Output
{
  "config": "<yaml string>",
  "context": "<file path>",
  "phase": "candidate_ready"
}
```

Listing C.7: Simplified system prompt for the Config Agent (A_{c}).

```
# Config Agent Input Context
dataset:
  name: cmapps
  fd_number: 2
  unit_col: unit
  os_cols: [os1, os2, os3]
  sensor_cols: [s1, s2, ..., s21]
  rul_clip: 125
  healthy_frac: 0.4

preprocessing:
  normalization:
    per_op_setting: true
    opc_mode:
      kmax: 6
  target_scaling: true

feature_engineering:
  feature_selection: [mutual_information, correlation]
  forbid: [tsfresh, flatten]

windowing:
  type: sliding
```

```yaml
  seq2val: true
  search_space:
    window_size: [50, 56, 64, 70]
    stride: [5, 8, 10]

# Seq2Val Model Search Space
seq2val_search:
  model: [gru, lstm]
  hidden_size: [64, 128, 256]
  num_layers: [1, 2, 3]
  dropout: [0.0, 0.2, 0.3]
  bidirectional: [false]
  optimizer:
    lr: [5e-4, 1e-3, 2e-3]
    weight_decay: [0.0, 1e-5, 1e-4]
  batch_size: [64, 128, 256]
  epochs: 60

objective:
  name: val_obj
  mode: min

# Prompt Agent Guidance (Consumed by Config Agent)
key_trends:
  - Seq2Val models (LSTM/GRU) perform best on C-MAPSS
  - Normalize per operating condition using OPC mode clustering (kmax = 6)
  - Apply RUL clipping (125) and retain a healthy fraction of ~0.4
  - Typical window size: 50-70, stride: 5-10
  - Prefer MI and correlation-based feature selection; avoid tsfresh/flatten
```

Listing C.8: Example guidance produced by the Prompt Agent (A_{p}) for the Config Agent (A_{c}); C-MAPSS FD002.

C.2 Example Configuration for C-MAPSS

```yaml
dataset:
  name: cmapps
  version: '0.9.5'

pipeline:
  type: { space: choice, values: [seq2val] }

criterion:
  kind: mse
  lam: 0.0
  reduction: mean

preprocessing:
  imputation:
    method: ffill_bfill
  denoising:
    method: none
    ignore: []
    group_by: none
    params: {}
  normalization:
    method: opc_mode
```

```yaml
    norm_type: standard
    os_cols: [os1, os2, os3]
    group_by: [unit]
    ignore: [unit]
    sensor_cols: auto
    kmax: 6
    min_level_frac: 0.02
    add_mode_onehot: false
  select_by_correlation:
    apply: true
    correlation_threshold: 0.9
    ignore: [unit]
  select_topk_mi:
    apply: true
    target_col: rul
    top_k: 16
    ignore: [unit]
    n_neighbors: 3
    random_state: 0
  resampling:
    apply: true
    group_by: unit
    keep_healthy_fraction: 0.4
    clip: 125
    rul_col: rul
    per_cycle: false
    cycle_col: cycle
  target_scaling:
    method: standard
    column: rul
    clip: 125
    drop: []
  cleanup:
    drop: []

feature_engineering:
  window:
    drop_last: false
    drop_first: false
    axis: 0
    group_by: unit
    size: { space: qrandint, low: 50, high: 70, q: 2 }
    stride: { space: qrandint, low: 5, high: 10, q: 1 }
    by_cycle: false
    fast: false

feature_extraction:
  model: { space: choice, values: [cnn-ae] }
  models:
    cnn-ae:
      latent_dim: { space: choice, values: [8, 16, 32] }
      hidden_ch: { space: choice, values: [32, 64, 128] }
      epochs: 100
      batch_size: { space: choice, values: [64, 128] }
      learning_rate: { space: choice, values: [0.0005, 0.001] }
      weight_decay: { space: choice, values: [0.0, 0.0001] }
      patience: 10
      scheduler: { space: choice, values: [none] }
      step_size: 20
```

```
      gamma: 0.5
      random_seed: 42

regression:
  model: { space: choice, values: [mlp_torch] }
  multi_window:
    apply: false
    m: 1
  models:
    mlp_torch:
      hidden: { space: choice, values: [64, 128, 256] }
      dropout: { space: choice, values: [0.0, 0.2, 0.3] }
      learning_rate: { space: choice, values: [0.0005, 0.001] }
      weight_decay: { space: choice, values: [0.0, 0.0001] }
      batch_size: { space: choice, values: [64, 128] }
      epochs: 120
      patience: 20
      scheduler: { space: choice, values: [steplr, cosine] }
      step_size: 20
      gamma: 0.5

seq2val:
  model: { space: choice, values: [cmapps_0_9_5_gru_seq2val_opc64, gru, lstm] }
  multi_window:
    apply: false
    m: 1
  trainer:
    epochs: 60
    patience: 10
    learning_rate: { space: choice, values: [0.0005, 0.001, 0.002] }
    weight_decay: { space: choice, values: [0.0, 0.00001, 0.0001] }
    batch_size: { space: choice, values: [64, 128, 256] }
    scheduler: { space: choice, values: [none, steplr, cosine] }
    step_size: 20
    gamma: 0.5
    t_max: 50
  models:
    cmapps_0_9_5_gru_seq2val_opc64:
      hidden_size: { space: choice, values: [64, 128, 256] }
      num_layers: { space: choice, values: [1, 2, 3] }
      dropout: { space: choice, values: [0.0, 0.2, 0.3] }
      bidirectional: false
      seed: 0
    gru:
      hidden: { space: choice, values: [64, 128, 256] }
      num_layers: { space: choice, values: [1, 2, 3] }
      dropout: { space: choice, values: [0.0, 0.2, 0.3] }
      bidirectional: false
    lstm:
      hidden: { space: choice, values: [64, 128, 256] }
      num_layers: { space: choice, values: [1, 2, 3] }
      fc_hidden: { space: choice, values: [32, 64, 128] }
      dropout: { space: choice, values: [0.0, 0.2, 0.3] }
      bidirectional: false
```

Listing C.9: Example YAML configuration with hyperparameter search space for the C-MAPSS FD002 dataset.

Appendix D

Evaluation Details

D.1 Model Architectures

Method	Dataset	Feature extractor	Regressor / Seq2Val	Windowing	Origin	RMSE
Agentic (Standalone)	FD001	raw	GRU (Seq2Val)	$L = 60$	G	17.80
Agentic (Standalone)	FD002	raw	BiGRU-MLP (Seq2Val)	$L = 50$	G	11.60
Agentic (Standalone)	FD003	raw	GRU (Seq2Val)	$L = 60$	G	13.96
Agentic (Standalone)	FD004	raw	GRU (Seq2Val)	$L = 60$	G	24.94
Agentic (Standalone)	FEMTO	Dilated CNN-AE	DeepNarrowMLP	$L = 2560$, $M = 20$	G	41.90
AutoML	FD001	raw	GRU (Seq2Val)	$L = 54$	P	13.00
AutoML	FD002	raw	GRU (Seq2Val)	$L = 55$	P	13.48
AutoML	FD003	raw	GRU (Seq2Val)	$L = 55$	P	11.93
AutoML	FD004	raw	GRU (Seq2Val)	$L = 96$	P	13.13
AutoML	FEMTO	raw	TCN (Seq2Val)	$L = 2560$, $M = 13$	P	31.43
Agentic (Hybrid)	FD001	raw	CNN-LSTM (Seq2Val)	$L = 65$	P	16.38
Agentic (Hybrid)	FD002	raw	GRU (Seq2Val)	$L = 70$	P	10.94
Agentic (Hybrid)	FD003	raw	GRU (Seq2Val)	$L = 58$	P	14.06
Agentic (Hybrid)	FD004	raw	GRU (Seq2Val)	$L = 98$	P	14.07
Agentic (Hybrid)	FEMTO	Dilated CNN-AE	XGBoost	$L = 2560$, $M = 20$	P	21.54

Table D.1: *Origin* indicates whether the architecture was predefined (P) or generated (G) by LLM. L denotes the window length and M the number of consecutive windows.

D.2 Error Metrics

D.2.1 NASA Score

The NASA scoring function, penalises late-life prediction errors more strongly than early predictions. For a set of N test instances, the score is defined as (Asif et al. 2022, p. 10):

$$\text{Score} = \sum_{j=1}^{N} s_j \tag{D.1}$$

with the individual penalty s_j given by:

$$s_j = \begin{cases} e^{-\frac{h_j}{13}} - 1, & \text{if } h_j < 0 \\ e^{\frac{h_j}{10}} - 1, & \text{if } h_j \geq 0 \end{cases} \tag{D.2}$$

where $h_j = \hat{y}_j - y_j$ denotes the prediction error for the j-th instance, with $\hat{y}_j$ being the predicted RUL and y_j the true RUL. Negative values correspond to early predictions, while positive values indicate late predictions.

D.2.2 Mean Absolute Error

The mean absolute error (MAE) measures the average magnitude of prediction errors, independent of their direction:

$$\text{MAE} = \frac{1}{N} \sum_{j=1}^{N} |\hat{y}_j - y_j| \tag{D.3}$$

D.2.3 Bias

Bias, also referred to as the mean bias error (MBE), quantifies the systematic tendency of a model to overestimate or underestimate the remaining useful life (RUL):

$$\text{Bias} = \frac{1}{N} \sum_{j=1}^{N} (\hat{y}_j - y_j) \tag{D.4}$$

A negative bias indicates systematic underestimation of RUL, while a positive bias corresponds to systematic overestimation.