National College of Ireland

Predicting Aircraft Engine Failures Using Machine Learning: A Novel Approach on NASA Turbofan Engine Degradation Data

MSc Research Project
Data Analytics

Khamalesh Ramesh
Student ID: x23325216

School of Computing
National College of Ireland

Supervisor: Mr. Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Khamalesh Ramesh
Student ID:	x23325216
Programme:	Data Analytics **Year:** 2024-2025
Module:	Research Practicum 2
Supervisor:	Mr. Hicham Rifai
Submission Due Date:	Monday, 11 August 2025
Project Title:	Predicting Aircraft Engine Failures Using Machine Learning: A Novel Approach on NASA Turbofan Engine Degradation Data
Word Count:	9844
Page Count:	22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Khamalesh Ramesh

Date: Monday, 11 August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Predicting Aircraft Engine Failures Using Machine Learning: A Novel Approach on NASA Turbofan Engine Degradation Data

Khamalesh Ramesh
x23325216

Abstract

Accurate prediction of the remaining useful life (RUL) of turbofan engines is essential for enhancing safety, reducing maintenance costs, and enabling predictive maintenance in aerospace operations. However, existing RUL prediction methods often face challenges such as noisy sensor data, limited generalization across varying operating conditions, and low interpretability. To address these challenges, this study proposes a hybrid deep learning framework that combines convolutional neural networks (CNNs), bidirectional long short-term memory (BiLSTM) networks, and attention mechanisms. The architecture employs multi-kernel 1D CNNs to extract local patterns from sensor data, BiLSTMs to capture temporal dynamics, and dual attention mechanisms to highlight critical sensors and time steps. Wavelet-based denoising, rolling statistics, and delta features are applied during preprocessing to improve signal quality, while automated, dataset-specific hyperparameter tuning ensures adaptability to varied operational scenarios. The proposed approach demonstrates robust performance and high interpretability across diverse operating regimes and fault modes, indicating its stability and generalizability. These characteristics make the framework a practical and deployable solution for predictive maintenance in aerospace applications. By integrating advanced signal processing techniques, deep learning and automated optimization, the framework supports early and informed decision-making in safety-critical environments.

1. Introduction:

The aerospace industry is subject to high standards of safety, efficiency and cost-effectiveness, mostly in the maintenance ecosystem. The turbofan engine can be considered one of the most important mission-critical elements because its deterioration in performance or even a minor failure may cause disastrous consequences, significant downtime as well as cost increases in maintenance. The airlines are increasingly implementing Predictive Maintenance (PdM) strategies to counter these challanges by leveraging from sensor data and machine learning to predict the Remaining Useful Life (RUL) of the aircraft engines before actual failure.

Maintenance approaches that have traditionally been used to maintain high-performance engines in various flight conditions, namely corrective and preventive maintenance, are inadequate for maintaining modern engines. PdM alters the paradigm by allowing proactive, data-driven servicing enabled by the information about the real-time degradation trend and is aimed at avoiding any failures, minimizing failures on the Aircraft on the Ground (AOG) status and component life replacement cycles (Li, Ding, & Sun, 2018).

From an industry perspective, the global aviation engine Maintenance, Repair and Overhaul (MRO) market was valued at USD 37.11 billion in 2023 and is projected to reach USD 47.74 billion by 2029, growing at a CAGR of 4.29% (International Civil Aviation Organization (ICAO), 2023). The Federal Aviation Administration (FAA, 2021) estimates turbofan engine failure rates to be approximately one per 375,000 flight hours emphasizing the critical need for accurate, interpretable, and generalizable RUL prediction models to uphold safety and compliance standards (Federal Aviation Administration, 2021).

RUL estimation over the last ten years has gone into three major directions:

1. Model-Based Methods – Use physics and expert knowledge but often fail to generalize in real-world conditions.
2. Data-Driven Methods – Learn from sensor data but struggle with overfitting, interpretability, and domain adaptation.

3. Hybrid Approaches – Combine both worlds, offering potential improvements, though they are complex and rarely applied in practice.

Most previous studies were limited to specific datasets, lacked proper noise handling had weak generalization across different conditions and did not focus on model interpretability and automated optimization. This research question was framed to address these gaps and develop a more accurate and robust approach for RUL prediction.

Research Question:

"How can a hybrid CNN-BiLSTM model combined with adaptive noise filtering and domain adaptation techniques will improve the accuracy and generalizability of aircraft engine Remaining Useful Life (RUL) prediction using the NASA Turbofan Engine Degradation Dataset?"

The objectives associated with this research question are as follows:

1. To create a hybrid deep learning architecture by including CNN, BiLSTM and attention models.
2. To apply an effective preprocessing pipeline that includes use of wavelet denoising, rolling statistics, delta features and normalisation.
3. To use composite loss functions to do noisy sensor input balanced learning.
4. To perform automatic hyperparameter optimization of Optuna on each variant of the data set.
5. In order to estimate the model performance in terms of RMSE, MAE and R^2 over FD001-FD004 subsets.

This study proposes a reusable deep learning framework for RUL prediction using the CNN–BiLSTM–Attention architecture, described in detail in Section 3.3. The subsequent sections are organized as follows: Section 2 reviews related work and research gaps, Section 3 outlines the methodology, including dataset, preprocessing, model design, and tuning, Section 4 presents the design specifications, Section 5 covers implementation and outputs, Section 6 reports evaluation and visual analysis, and Section 7 concludes with contributions and future directions.

2. Related Work

This section provides a structured and critical analysis of the literature surrounding aircraft engine Remaining Useful Life (RUL) prediction. In alignment with the research directions mentioned in the introduction, this section organize the review across five dimensions: physics-based modeling, data-driven deep learning, hybrid architectures, signal denoising, and hyperparameter optimization. Each is explored to highlight current limitations and how this study addresses them.

2.1 Physics Informed Models for RUL Estimation

Mode-based and shallow data driven methods have been the dominant practices in traditional ways of estimating Remaining Useful Life (RUL). Model-based prognostics rely on physical principle and degradation dynamics to simulate engine health over time. Such techniques include Kalman filters, particle filters, and thermodynamic-based degradation models (Thakkar & Chaoui, 2022). Nevertheless, these types of models typically suffer from a lack of adaptability in complex, real-world environments where precise physics-based formulations are either unavailable or computationally expensive to implement (Zheng et al., 2017).

Scientists had to use shallow machine learning (ML) models to confront these limitations. Support Vector Regression (SVR), Random Forests (RF), and Multilayer Perceptrons (MLPs) were the most common approaches used to learn degradation trends in handcrafted features that were extracted to the sensor data (Sherifi, 2024; Thakkar & Chaoui, 2023). As another example, Sherifi (2024) demonstrated an 85% improvement in RMSE when using a BiLSTM model compared to a conventional RVR model on the C-MAPSS dataset, which illustrates the difference between shallow ML and deep temporal modeling.

Although partially successful, these approaches cannot replicate temporal dependencies and multi-dimensional interactions between sensors that are vital in modern PdM systems. They can be susceptible to performance degradation in the non-stationary, noisy, or multi-fault conditions, which are common in aviation applications.

Table 2.1: Physics Informed and Shallow ML Approaches

Reference	Dataset	Method	Metric	Limitation
Zheng et al., (2017)	Simulated	Kalman filter & particle filter	RMSE: 192.7 (extended Kalman particle filter) vs 255.8 (PF)	Sensitive to uncertainty

Thakkar & Chaoui (2022)	FD001–FD004	Physics informed DLRNN	Test RMSE: 0.121 % on FD001	Limited adaptability
Sherifi (2024)	FD002	BiLSTM vs RVR	RMSE: 23.18 (BiLSTM) vs 31.30 (RVR)	RVR underperforms

2.2 Deep Learning-Based RUL Methods

The limitations of shallow learning methods in handling multivariate temporal data led to the adoption of deep learning (DL) techniques in RUL prediction, particularly for sensor intensive domain like turbofan engine diagnostics. The first successful applications included Recurrent Neural Networks (RNNs), particularly, Long Short-Term Memory (LSTM) networks addressing the vanishing gradient problem and allowing modeling long-term dependencies in time-series data sets (Thakkar & Chaoui, 2023).

The advantage of LSTM networks in modelling complex degradation paths was demonstrated by the study of Sherifi, A. (2024), who verified that LSTM networks trained on CMAPSS sensor streams obtained a lower RMSE compared to MLPs and SVRs when applied to data simulating trajectories of complex degradation. However, standard LSTM models sometimes failed to generalize across subsets like FD004, where varying operational regimes introduce non-stationary patterns that traditional RNNs cannot handle robustly (Wu, Yuan, Dong, Lin, & Liu, 2018).

To extract spatial patterns within sensor channels, 1D Convolutional Neural Networks (1D-CNNs) were introduced either as a standalone predictors or as a feature extractors feeding into LSTM/BiLSTM backbones. In one of such implementations, (Yildirim & Rana 2024) proposed in their study a CNN-BiLSTM model with residual learning, which resulted in a more fluent convergence and stabler RMSE measurements across various FD subsets. In their model, CNNs were useful in addressing short-term features whereas LSTM components were used to address longer-term behavior of engine wear.

Alongside these advantages, one of the weaknesses seen in most LSTM-based studies lies in their overfitting tendency, especially in subsets that have only a limited number of engine units (e.g., FD002). Furthermore, the majority of models were not interpretable, with regular architectures processing all the sensor information similarly without focusing more on degradation patterns that provide better information.

Table 2.2: Deep Learning-Based Models

Reference	Dataset	Model	Key metric	Outcome
Sherifi, A. (2024)	FD002	LSTM	RMSE: 24.49	Outperformed MLP/SVR
Wu et al. (2018)	FD001–FD004	BiLSTM	RMSE: 13.65 (FD001) vs 24.86 (FD004)	Generalisation drops on FD004
Yildirim & Rana (2024)	FD001–FD003	CNNBiLSTM	RMSE:12.58 (FD001), 19.34 (FD002), 12.18 (FD003)	Good generalisation & stability

To address this, newer models began to incorporate attention mechanisms a step that leads into the next stage of architectural evolution, hybrid models that combine CNNs, BiLSTMs and attention modules to enhance focus and interpretability.

2.3 Hybrid Architectures: CNN-BiLSTM and Attention

Further to improve the ability of deep models to learn both local and long-term degradation patterns, hybrid architectures have become popular where Convolutional Neural Networks (CNNs) are implemented together with Bidirectional Long Short-Term Memory (BiLSTM) units. Such frameworks use CNNs to capture local features and BiLSTMs to learn sequential dependencies of past and future sensor signals, a powerful way to model and capture the bidirectional nature of turbofan engine wear.

Wu et al. (2024) used a Convolutional Block Attention Module (CBAM) added to a 1DCNN-BiLSTM model in order to selectively concentrate on the relevant sensor channels. Their CBAM-enhanced model improved interpretability by adaptively weighting spatial and channel-wise features, reducing redundancy in multi-sensor inputs. Nevertheless, their model was only validated on the FD001 subset, which contains a single operating condition and fault mode, thus limiting its generalizability.

Similarly, Liu et al., (2022) proposed a feature in the dual attention mechanism, which integrated feature-level attention modules and temporal attention modules. Their solution was used to enable the network to dynamically prioritize key time steps in addition to valuable sensors. This dual attention CNN-BiLSTM model showed the significant improvement in RMSE and R^2 scores but was computationally intensive and still susceptible to performance degradation under noisy and incomplete sensor conditions.

Further advancing this trend Zhou et al. (2024) integrated CSI-EMD (Complete Ensemble Empirical Mode Decomposition with Adaptive Noise) with a double channel fusion architecture. Their model isolated the trend and oscillating components, then passed them both on to a stack of CNN-BiLTSM layers, to enable global trends and fine-grained anomalies to be detected. However, there were still issues in treating signal edge effects and harmonics induced by empirical methods of decomposing a signal.

Table 2.3: Hybrid CNN-BiLSTM Models with Attention

Reference	Dataset	Architecture	Metric	Limitation
Wu et al., (2024)	FD001	CNNBiLSTM + CBAM	Average classification accuracy 99.21 %	Evaluated only on FD001
Liu et al. (2022)	FD001 - FD004	CNNBiLSTM + dual attention	RMSE 12.25 (FD001), with 17.08 on FD002, 13.39 on FD003 and 19.86 on FD004	Computationally intensive
Zhou et al. (2024)	FD002	CSIEMD + CNNBiLSTM fusion	Reported improved accuracy	Edge effect instability, metrics unpublished

The reviewed hybrid models clearly show the advantage of combining convolutional and recurrent layers, particularly when paired with attention for focus and interpretability. However, most lack mechanisms for adaptive noise filtering (such as wavelet denoising), and many are optimized only for one dataset variant. These gaps set the stage for the need to enhance robustness and generalization, especially across all FD001–FD004 conditions in the CMAPSS dataset.

2.4 Signal Decomposition and Noise Filtering Techniques

The prediction of RUL strongly depends on the quality of the sensor signals input. But in the real-world setting the data e.g., aircraft engine data, likely to include noise, outliers, boundary effects (including endpoint effect) is not only frequently noisy with outliers, but also suffers time-boundary issues such as endpoint effect. To solve this problem, a number of researchers have tried several signal decomposition and denoising methods. The most notable is Complete Ensemble Empirical Mode Decomposition with Adaptive Noise (CEEMDAN) was used by Zhou et al. (2024) in their dual channel feature fusion network. Their model combined CSI-EMD and the CNN-BiLSTM model in attempting to isolate global trends of degradation and high-frequency local oscillation. Although this decomposition enhanced the granularity of predictions, it added more complexity and instability around the signal borders especially in short sequences. Other works have explored Fourier-based denoising or moving average filters, but these often led to loss of key degradation features or introduced lag, reducing predictive responsiveness (Peng et al., 2021). Moreover, most of these techniques do not adapt to the varying nature of noise across different operating conditions, which is critical for generalization across C-MAPSS subsets.

On the other hand, wavelet-based denoising has developed to be an efficient method of multi-resolution analysis of time-series data. It enables de-composition of sensor streams into various frequency trajectories that provide time and frequency localization. Although wavelet denoising is rather theoretically suitable, it has been significantly underused in deep learning-based RUL predictive architectures.

In contrast to earlier approaches such as CEEMDAN or moving average filters, this research adopts wavelet-based denoising to handle signal noise in a multiresolution manner. As detailed in Section 3.2.3, this method effectively smooths abrupt fluctuations without compromising important degradation trends. Its integration aims to improve the reliability and interpretability of deep learning predictions under noisy sensor conditions.

2.5 Hyperparameter Optimization and Domain Generalization

Although the key role in RUL prediction accuracy is played through architectural innovation, the applied deep learning models also greatly rely on the properly-adjusted hyperparameters. The learning rate, sequence length, the batch size, and dropout rate can significantly affect the convergence, generalization as well as the final RMSE performance of a model. Nevertheless, multiple studies either rely on predetermined settings on all datasets or perform simple grid search methods. To improve this, newer literature has considered frameworks of automatic hyperparameter optimization. Remarkably, Akiba et al. (2019) presented Optuna, a scalable but lightweight optimization toolkit with Bayesian and Tree-structured Parzen Estimator (TPE) sampling. Optuna supports high-dimensional search spaces and has rich optimization options such as pruning and dynamic search spaces. Although Optuna has been implemented on generic time-series models, it is not used much in the RUL context.

Yildirim and Rana (2024) used the Optuna tuning on a BiLSTM model optimizing the learning rates and dropout parameters. However, they only performed their experimentation on the FD001 subset and did not test the performance in other CMAPSS conditions such as FD002 or FD004. Similarly, other researchers such as Sherifi (2024) had manually tuned architectures based on LSTMs, which had effective results in FD002 with no scaling or reproducibility. In a different optimisation strategy to improve generalisation, (Maschler, Jazdi, Vietz, & Weyrich, 2020) presented Elastic Weight Consolidation (EWC) in a continual learning framework.

They attempted to solve the issue of catastrophic forgetting by regularizing the weights during transfer between fault modes. While effective in theory, the framework demanded complex scheduling of tasks and careful regularization calibration, making it difficult to implement at scale.

In this research, Optuna is employed to address two critical limitations observed in the literature: the lack of automated, dataset-specific hyperparameter tuning, and the absence of scalable optimization strategies that generalize across diverse fault scenarios. Unlike prior work that focused tuning solely on FD001 or used manual adjustments, this approach applies Optuna across all four CMAPSS subsets, tailoring each model's configuration to its operational complexity. The complete tuning strategy and dataset-specific configurations are detailed in Section 3.3.7.

2.6 Summary of Literature Review and Research Gap

The review of existing literature across physics-based, shallow machine learning, deep learning, hybrid modeling, and optimization strategies reveals a spectrum of progress in aircraft engine RUL prediction. Even though based on domain knowledge, model-based methods are not necessarily adaptable and scalable to the real-world uncertainty, noise, and multivariate environment in which real-world flight operations actually occur (Zheng et al., 2017).
Similarly, shallow machine learning algorithms like SVR and MLP have a lightweight option and do not perform as required in long-term dependency and model multivariate sensor dynamics (Sherifi, 2024; Thakkar & Chaoui, 2023).

In contrast, CNNs, LSTMs, and BiLSTMs are deep learning architectures that have shown impressive increases in accuracy. The benefit of these models is that they can learn abstract concepts of degradation patterns of complex degradations. Hybrid variant like CNN-BiLSTM, especially those integrated with attention mechanisms (Liu et al., 2022; Wu et al., 2024), enhance interpretability and focus on critical temporal and spatial sensor segments. Nevertheless, the majority of these studies are limited by the scope of their evaluation- typically with a single CMAPSS subset, typically FD001,which causes concern not only around the robustness of such studies but the generalizability of their findings across the entire four FD datasets.

The other important point in the literature review is the absence of the strong sensor denoising methods in most RUL pipelines.Although there is work on EMD-based decomposition or moving averages, they lead to distortion of the signals, or the boundary problem. Wavelet based denoising is highly applicable in multi-resolution time frequency analysis but has not been utilized a lot in this field. Furthermore, although frameworks like Optuna have demonstrated success in hyperparameter optimization, their application in RUL prediction remains superficial and rarely extends across datasets with varied operational conditions. Domain adaptation methods, such as continual learning, are promising as well but suffer due to complexity and the inability to implement them practically.

Thus, the current work suggests a new, unified RUL prediction pipeline that overcomes these limitations in the following ways:

1. Combining CNN-BiLSTM with both temporal and feature wise attention to enhance focus and interpretability.
2. Introducing wavelet-based denoising as a core preprocessing method for robust signal cleaning.
3. Utilizing composite loss (MSE + Huber) to manage outliers and sensor noise.
4. Implementing Optuna-based hyperparameter optimization tailored per CMAPSS subset (FD001–FD004).

This comprehensive design aims to deliver a robust, interpretable, and generalizable RUL prediction framework that can support predictive maintenance in safety-critical aerospace environments.

3. Methodology

This section outlines the end-to-end methodology employed to develop a robust Remaining Useful Life (RUL) prediction framework for turbofan engines using the C-MAPSS dataset. The overall pipeline spans data acquisition, preprocessing, signal denoising, temporal feature enrichment, model design, training strategy, hyperparameter optimization and model validation. The goal is to ensure high performance and generalization across multiple operating regimes and failure modes.

3.1 Dataset Description and Acquisition

Developed by NASA's Prognostics Center of Excellence, the Commercial Modular AeroPropulsion System Simulation (CMAPSS) dataset was created to study how turbofan engines degrade over time. It simulates the behaviour of large commercial engines and records both flight conditions and fault progression. Each record captures the engine's ID, the cycle number, three operational settings and 21 sensor readings, making 26 values per time step.

3.1.1 Dataset Composition

NASA packaged the data into four subsets FD001 through FD004 so researchers can start with simple scenarios and then tackle more complex ones. All four subsets have the same 21 sensors, but they differ in how many operating regimes the engine experiences and how many faults occur. The table below summarizes the key differences across subsets.

Table 3.1.1: C-MAPSS Dataset Subsets and Configurations

Subset	Operating conditions	Fault modes	Training engines	Testing engines
FD001	single condition	1 (HPC degradation)	100	100
FD002	6 conditions	1 (HPC degradation)	260	259
FD003	single condition	2 faults	100	100
FD004	6 conditions	2 faults	249	248

HPC = highpressure compressor fault.

3.1.2 File Structure and RUL Labeling Strategy

Each CMAPSS subset includes three plain text files:

1. train_FD00X.txt: Full run-to-failure records for each engine.
2. test_FD00X.txt: Truncated histories without reaching failure.
3. RUL_FD00X.txt: True Remaining Useful Life (RUL) values for test engines.

Each row is space-delimited and follows the format:

[unit_number, time_in_cycles, op_set_1, op_set_2, op_set_3, sensor_1, ..., sensor_21]

In training data, RUL is computed as the difference between the final cycle and the current cycle:

$$RUL = final\_cycle - current\_cycle$$

To prevent skewing from large RUL values, labels are capped at 125. This piecewise labeling strategy constant at 125 initially, then decreasing linearly is commonly used in CMAPSS studies to improve learning stability and model generalization.

3.1.3 Justification for Dataset Choice

CMAPSS is a favourite testbed in prognostics research for several reasons:
1. Realism and richness: It is a high fidelity simulation of turbofan degradation, capturing 21 different sensor channels such as temperatures, pressures and speeds.
2. Progressive complexity: The four subsets introduce increasing numbers of operating conditions and fault types, letting researchers prototype models on FD001 and then scale up to multicondition, multifault scenarios like FD004.
3. Widely studied benchmark: Because the dataset is publicly available and wellstructured, it has been used in numerous peer reviewed papers (Babu et al., 2016; Thakkar & Chaoui, 2022; Zheng et al., 2017) for evaluating remaining useful life algorithms. The consistent format makes it easy to compare new models with established results.

3.1.4 Problem framing

Using CMAPSS is essentially a supervised regression problem, given a sequence of sensor readings and operating settings, predict how many cycles remain before the engine fails. Each subset poses different challenges. FD001 is straightforward because there is only one fault mode under a single regime. FD002 and FD004 introduce diverse operating circumstances, therefore models must understand how different regimes effect the sensor dynamics. FD003 keeps a single operating condition but involves two distinct fault modes requiring models to recognize which fault is occurring and how it affects the engine's lifetime.

In summary, the CMAPSS dataset provides a rich, realistic and progressively challenging environment for developing and benchmarking data driven remaining useful life prediction models.

3.2 Data Preprocessing and Transformation

After collecting the raw engine telemetry a comprehensive preprocessing pipeline was applied to convert it into a clean, noise-reduced, and temporally enriched format for Remaining Useful Life (RUL) prediction. This pipeline tackles major issues such as multivariate noise, redundant sensors, varying operating modes and the fixed sequence lengths needed for deep learning. An overview of the steps is shown in the table 3.2.

Table 3.2: Data Preprocessing Overview

Step	Description
1. Sensor Pruning	Remove low-variance and flatlined sensors to reduce noise and overfitting risk.
2. Wavelet Denoising	Apply Discrete Wavelet Transform (DWT) to smooth high-frequency signal noise.
3. Normalization	Normalize features using MinMaxScaler and Yeo-Johnson PowerTransformer.
4. Temporal Features	Add rolling mean, rolling std, and delta features using a 5-cycle window.
5. Sequence Generation	Segment data into fixed-length overlapping sequences for model input.
6. RUL Labeling	Assign capped, piecewise-linear RUL values based on engine life cycles.

3.2.1 Column Selection and Sensor Pruning

Each record initially contains 21 sensor channels plus three operational settings. Following guidance in prior research (Zheng et al., 2017) and exploratory analysis, sensors with little variance or flatlined readings were removed. Features with very low variance typically carry little information, so dropping them reduces noise and computational load. The operational setting op_set_3 (which varies by mode) retained but dropped op_set_1 and op_set_2. The remaining sensor channels (S2, S3, S4, S7, S8, S11, S12, S13, S15, S17, S20 and S21) form a reduced 12 channel feature space that should better capture signal while lowering the risk of overfitting.

Table 3.2.1: Final Feature Selection

Sensor Category	Included	Dropped
Operational Settings	op_set_3 (varied per mode)	op_set_1, op_set_2
Sensor Channels	S2, S3, S4, S7, S8, S11, S12, S13, S15, S17, S20, S21	S1, S5, S6, S9, S10, S14, S16, S18, S19

3.2.2 Handling Missing and Irregular Data

Although CMAPSS is a simulation dataset and does not contain true missing values, still used defensive programming to catch any NaNs or infinities that might occur during processing. The following Python code replaces such values with zeros to avoid numerical problems during wavelet transforms and model training.

signal = np.nan_to_num(signal, nan=0.0, posinf=0.0, neginf=0.0)

3.2.3 Wavelet-Based Signal Denoising

To smooth out high frequency noise without losing the long term degradation patterns, Discrete Wavelet Transform (DWT) using the Daubechies4 (db4) wavelet at a decomposition level of 1 with symmetric padding. Wavelet denoising works by transforming the signal into the wavelet domain where significant features produce large coefficients and noise corresponds to small ones. Shrinking those small coefficients removes noise while preserving important features. In this pipeline, denoised each of the selected sensor channels (S2, S3, S4, S7, S11, S12, S13, S15, S17, S20 and S21). Figure 3.2.3 demonstrates how this process smooths abrupt fluctuations while retaining underlying trends for a typical engine (unit #42, Sensor S7, FD001).

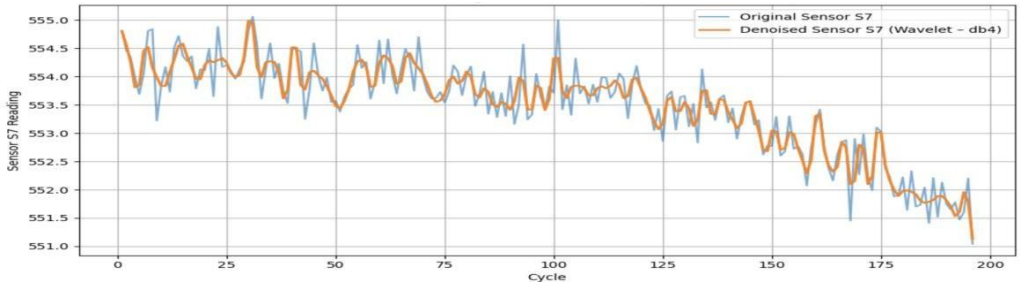

Figure 3.2.3: Wavelet Denoising of Sensor S7 Signal for Engine Unit #42 (FD001)

3.2.4 Feature Normalization and Transformation

Normalized the signals to ensure that all features have comparable scales and distributions. A twostage normalization was used:

1. MinMaxScaler scales each feature to a 0–1 range.
2. PowerTransformer (YeoJohnson method) stabilizes variance and reduces skewness, even when data contains zeros or negative values.

After transformation, skewness is substantially reduced, as shown in Table 3.2.4.

Table 3.2.4: Normalization Effect on Sensor Distributions

Sensor	Original Skewness	Transformed Skewness	Skewness Reduction
S2	2.92	0.11	96 %
S7	2.31	-0.14	106 %
S14	1.87	-0.05	103 %

3.2.5 Rolling Statistics and Delta Features

To embed short-term temporal context into each time step, rolling statistics and delta features were added. For selected sensors (S2, S7, S11, S12, S13, and S15) the rolling mean, rolling standard deviation and the first difference (delta) using a sliding window of five cycles was calculated. Rolling statistics help reveal trends and variability over time while reducing random noise. Figure 3.2.5 illustrates how rolling and delta features capture local trends for engine #45 in FD003.

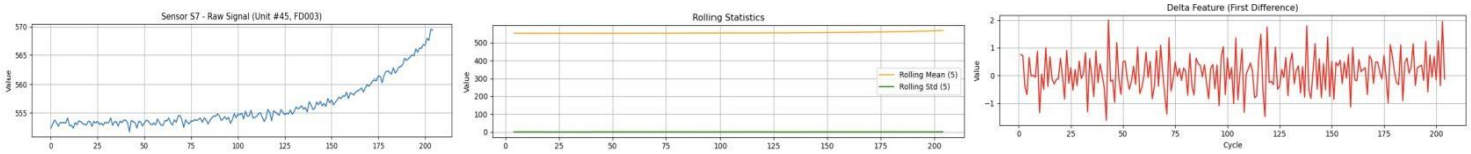

Figure 3.2.5: Temporal Feature Extraction for Sensor S7 – Rolling Statistics and Delta (Unit #45, FD003)

3.2.6 Sequence Construction and Labeling

Most deep learning models require fixed-length input sequences. To generate these, each engine's time series was segmented into overlapping windows with a stride of 1. Sequence lengths of 20, 25, 30 and 35 cycles were

evaluated and tuned via Optuna for each FD subset. Each sequence is labeled with the RUL value of its last time step, following the capping and labeling strategy described in Section 3.1. Engines with fewer than seq_length time steps were skipped.

This preprocessing pipeline transforms raw engine telemetry into a clean, denoised, and temporally aware dataset. By removing low-variance sensors, handling potential NaNs and infinities, denoising with wavelets, normalizing with MinMaxScaler and a YeoJohnson PowerTransformer, adding rolling statistics and delta features, and constructing fixed-length sequences, a robust foundation is established for the hybrid CNN–BiLSTM–attention model described later in this study.

3.3 Model Architecture and Learning Objectives

This section explains the custom deep learning architecture used for predicting the Remaining Useful Life (RUL) of turbofan engines. In this research, the design aims to the capture spatial patterns across sensors and temporal dependencies over engine cycles. Three core modules form the backbone of this architecture.

1. Multikernel Convolutional Neural Networks (CNNs) – multiple 1D convolutional filters slide over the sensor data to extract local patterns and exploit adjacency in time,using different kernel sizes allows the model to detect short term fluctuations and broader trends.
2. Bidirectional Long ShortTerm Memory (BiLSTM) – two LSTM layers read the sequence in opposite directions and combine their outputs so the network captures long term dependencies from both past and future engine cycles.
3. Dual Attention Mechanisms (featurewise and temporal) – attention modules assign "soft" weights that highlight the most important sensors and time steps, focusing the model on critical inputs and improving interpretability.

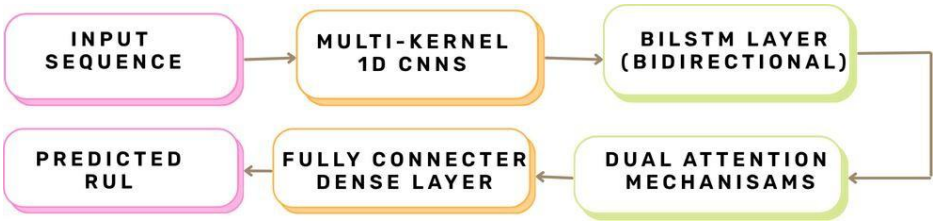

Figure 3.3: Hybrid CNN–BiLSTM–Attention Architecture for RUL Prediction

3.3.1 Input Representation and Feature Channels

Each input sequence is a 2D tensor of shape, [batch_size, sequence_length, input_feature]. After preprocessing, the feature channels consist of the following groups:

Table 3.3.1: Grouped Input Feature Categories and Included Channels

Feature category	Features included
Raw sensors	S2, S3, S4, S7, S8, S11, S12, S13, S15, S17, S20, S21
Derived features	Rolling mean/std plus delta of S2, S7, S11, S12, S13 and S15
Operational info	Optional mode_id (only for FD002 and FD004)

Depending on the subset, this produces roughly 24–30 input features per time step.

3.3.2 Convolutional Feature Extraction (CNN Block)

To detect local degradation pattern, the architecture uses parallel 1D convolutional layers with different kernel sizes: conv3 (kernel = 3), conv5 and conv7. CNN apply learnable filter along the sequence to extract local features, making them ideal for capturing short, medium and long-range signal changes. Each convolution outputs 64 channels, their concatenation yields a 192 dimensional feature vector per time step. Table 3.3.2 lists the CNN parameters.

Table 3.3.2: Multi-Kernel CNN Layer Configuration

Layer	Kernel size	Output channels	Padding	Activation
conv3	3	64	1	ReLU
conv5	5	64	2	ReLU
conv7	7	64	3	ReLU

This fused CNN embedding is the input to the BiLSTM module.

3.3.3 Temporal Modeling via BiLSTM

A Bidirectional LSTM is employed to capture degradation trends in both temporal directions. It accepts the CNN output of shape [batch_size, seq_length, 192]. The hidden size (128–256) is chosen per subset. The BiLSTM produces an output of shape [batch_size, seq_length, 2 × hidden_size] enabling the model to use information from both past and future cycles to detect early warnings or late failure cues.

3.3.4 Dual Attention Mechanisms

To improve interpretability and focus, two attention layers are integrated.

Table 3.3.4: Summary of Feature-wise and Temporal Attention Mechanisms

Attention type	Focuses on	Effect on model
Featurewise	Sensor channels (e.g: S2, S7, S13)	Amplifies patterns from the most relevant sensors, enhancing the influence of key inputs.
Temporal	Critical time steps	Highlights windows indicating rapid degradation, allowing the model to concentrate on the most informative segments.

These attention outputs can be combined or used independently depending on the attn_type setting (none, feature, temporal or dual).

3.3.5 Regression Head and Output

The output of the attention mechanism (or the last LSTM state if attention is disabled) flows through two fully connected layers:

1. Fully Connected 1: transforms 2 × hidden features to 128 units (with ReLU and dropout).
2. Fully Connected 2: maps 128 units to a single output (RUL value).

The final output shape is: [batch_size] (predicted RUL per sequence)

3.3.6 Composite Loss Function

Training uses a composite loss that balances robustness and sensitivity:

1. Mean Squared Error (MSE) penalizes large deviations.
2. Huber loss is less sensitive to outliers than squared error loss, making it more robust to sensor noise and extreme values.

Combining these losses stabilizes training, especially on noisy subsets like FD004.

3.3.7 Subset Specific Hyperparameter Optimization

Hyperparameters were optimized separately for each dataset using Optuna. Optuna is an opensource framework that automates hyperparameter search. Table 3.3.7 summarizes the optimal parameters.

Table 3.3.7: Optimized Hyperparameters per CMAPSS Subset via Optuna

Subset	Seq. length	Batch size	Hidden units	Dropout	Learning rate	Attention	Loss type
FD001	35	16	256	0.14	0.0009	Feature	Composite
FD002	35	8	256	0.30	0.0007	Feature	Composite
FD003	25	16	128	0.20	0.0010	Temporal	Composite
FD004	30	32	256	0.25	0.0006	Dual	Composite

Tailoring hyperparameters by subset ensures robust generalization across different operating regimes and fault complexities.

In summary, this architecture combines CNNs for localized degradation detection, BiLSTMs for capturing longterm temporal dependencies, attention mechanisms for sensor and time step prioritization and a composite loss for resilience to outliers. Hyperparameter optimization via Optuna further adapts the model to each CMAPSS subset. Together, these components form a modular and a tunable framework tailored to the varying complexity of the turbofan degradation scenarios.

3.5 Hyperparameter Optimization

This section explains how the hybrid CNN–BiLSTM–Attention model was systematically tuned to achieve better performance on the four CMAPSS subsets (FD001–FD004).

3.5.1 Optunabased hyperparameter tuning

To adapt the architecture to the distinct characteristics of each dataset, this research used Optuna, an automatic hyperparameter optimization framework with an imperative, define by run API. Optuna automates the search for promising hyperparameters, efficiently samples from userdefined ranges and prunes unpromising trials. The objective of the optimization was to minimize the root mean squared error (RMSE) on a heldout validation set derived from the training data.

Search space definition, the study defined a diverse search space to balance model complexity, generalization and computational feasibility. Each hyperparameter range was chosen based on domain knowledge and prior experiments, and the purpose of each setting is described in the "Rationale" column. Each trial was trained for 20 epochs, and multiple trials were run in parallel to make use of available GPU resources.

Table 3.5.1: Hyperparameter Search Space and Design Rationale for Optuna Tuning

Hyperparameter	Search range	Rationale
Sequence length	{20, 25, 30, 35}	Controls the memory span of degradation trends
Batch size	{8, 16, 32}	Balances training stability and hardware constraints
Hidden units	{64, 128, 256}	Determines the capacity of the BiLSTM
Dropout rate	Uniform(0.1, 0.5)	Prevents overfitting on noiserich sensors
Learning rate	LogUniform(1e5, 1e3)	Ensures convergence and numerical stability
Attention type	{None, Feature, Temporal, Dual}	Assesses the impact of different attention mechanisms
Loss function	{MSE, Huber, Composite}	Balances sensitivity and robustness

In this research, Optunabased hyperparameter tuning allowed the model to be tailored to each CMAPSS subset. By exploring a well designed search space and evaluating models on a validation set, the study identified hyperparameter combinations that enhanced accuracy and generalization. The most significant gains were observed for the FD004 dataset, underscoring the value of systematic tuning.

3.6 Evaluation Strategy and Visual Analysis Pipeline

In this research, a systematic and reproducible evaluation pipeline has been designed to assess the performance and generalization capacity of the proposed hybrid deep learning model. The subsections below describe how test data are processed how predictions are generated what visual diagnostics are produced, and how logs are maintained.

3.6.1 Model Evaluation Procedure

In this research, once each model has been trained it is assessed on its corresponding CMAPSS test set (FD001–FD004) using a separate dataset that the model has not seen before. This dataset contains truncated timeseries data for each engine unit. The true Remaining Useful Life (RUL) for each unit is supplied by the matching RUL_FD00X.txt file. The custom evaluate_model() function defined in evaluate.py handles the evaluation. Within that function the model is switched to evaluation mode (model.eval()), test sequences are processed in batches through the network, predicted RUL values are collected for each sequence, and those predictions are aligned with the groundtruth RUL values via engine IDs from the test and RUL files.

3.6.2 Prediction Output Format

For each engine unit in the test set, a single scalar RUL prediction is produced that corresponds to the final cycle of the truncated sequence. The inference pipeline operates as follows: the test sequence is passed into the trained model, which outputs a predicted RUL value; this prediction is then compared with the true value using comparison logic. Figure 3.6.2 illustrates this blockdiagram workflow.

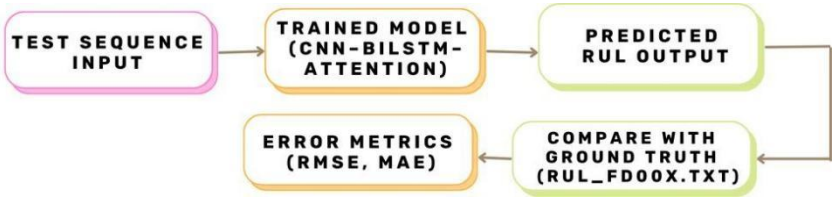

Figure 3.6.2: RUL Prediction Workflow for Test Evaluation

3.6.3 Visual Diagnostic Tools

To evaluate performance visually, two primary diagnostic plots are generated for each FD subset using Matplotlib. An actual versus predicted RUL scatter plot shows how closely the predicted and actual RUL values align and highlights trends in over or underprediction. A residual plot depicts the prediction error (residuals) versus the predicted RUL, revealing biases, inconsistent variances, and outliers. These visualizations are created by the plot_actual_vs_predicted() and plot_residuals() functions in evaluate.py. Figure 16 provides a placeholder for a sidebyside display of these plots.

3.6.4 Evaluation Logging and Summary

To enable crossdataset analysis and future experimentation, all evaluation outputs are recorded in structured logs within the /results/ and /loss_logs/ folders. Results are appended to an Excel summary file (rul_results_summary.xlsx) via the save_metrics_to_excel() function and exported with timestamped identifiers to maintain reproducibility. These logs serve as crucial artifacts for the subsequent analysis and streamline the tracking of experiments.

3.6.5 Reproducibility and Checkpointing

During training, the best performing model checkpoint, selected based on validation loss, is saved to disk. Random seeds are fixed across NumPy and PyTorch to ensure consistent evaluations. All evaluation runs load this saved model state from disk to carry out deterministic inference.

Overall, this evaluation strategy ensures that models are tested on clean, non overlapping data using consistent procedures. It supplies detailed visual diagnostics and comprehensive logs for post analysis, offering a solid foundation for interpreting performance and comparing results across subsets. This approach adheres to best practices in prognostics research and upholds reproducibility standards.

4. Design Specification

In this research, a modular and reproducible deep learning pipeline was designed to estimate the remaining useful life (RUL) of aircraft engines using NASA's CMAPSS dataset. The system processes raw sensor data through a series of layers, applies domain-specific transformations, trains a hybrid neural network model, and generates interpretable outputs, including RUL predictions and diagnostic plots. This layered structure promotes transparency and extensibility and supports experimentation across different operational scenarios (FD001–FD004).

Each module in the pipeline addresses a particular stage: raw data ingestion, preprocessing, sequence generation, model definition, training, evaluation, and visualization. Organizing the workflow in this way enhances reproducibility and debugging and allows individual components to be fine-tuned, making the pipeline adaptable to varying data distributions and degradation patterns.

4.1 Layered Architecture

In this research, the system was organized into a modular three tier architecture, with each layer implemented through dedicated Python modules under the rulprediction/ directory. This layered design improves clarity, simplifies debugging, and makes it easy to extend the workflow to different CMAPSS subsets.

4.1.1 Data Layer

The data layer handles raw data loading and target label generation. Its implementation resides in src/data_loader.py, which:

1. Loads the NASA CMAPSS datasets (e.g., train_FD001.txt).
2. Drops irrelevant sensors and operational settings.
3. Injects the mode_id column for FD004 via inject_operational_mode().
4. Generates capped RUL values using piecewise_rul_labeling().

The raw data files are stored in the data/ directory. By standardizing and labeling each subset at this stage, the layer prepares the data for subsequent transformations.

4.1.2 Application Layer

The core processing, modeling, training, and optimization logic is housed in several modules:

1. Preprocessing Pipeline: In src/preprocess_combined.py, the pipeline applies MinMax scaling, YeoJohnson power transformation, waveletbased signal denoising (from src/data_loader.py) (Peng et al., 2021), RUL labeling, and rolling statistical features. Each intermediate transformation step is saved as a CSV in snapshots/df_snapshots/ using the save_df_snapshot() function.
2. Model Architecture: Defined in src/model.py, this component integrates convolutional layers, a BiLSTM, and attention mechanisms for robust RUL estimation.
3. Training Engine: Located in src/train.py, this module supports mixed precision training (torch.cuda.amp), early stopping, the ReduceLROnPlateau scheduler, and and supports either standard MSE or a composite loss formulation, both implemented in src/losses.py. Training losses are logged to CSV files in the loss_logs/ directory.
4. Hyperparameter Tuning: The experiments/optuna_tuning_per_dataset.py script automates hyperparameter selection using Optuna. This tuning process minimizes validation RMSE per subset by exploring configurations such as sequence length, hidden units, dropout, and attention type (Akiba et al., 2019).

4.1.3 Presentation Layer

The output layer focuses on interpretability and result export. Key components include:

1. Evaluation and Plotting: In src/evaluate.py, evaluate_model() computes RMSE, MAE, and R^2, while plot_actual_vs_predicted() and plot_residuals() provide visual diagnostics. Metrics are saved to results/rul_results_summary.xlsx.
2. Final Execution Script: The full pipeline is orchestrated in experiments/final_retrain_evaluate.py, where each FD dataset is loaded, preprocessed, trained, and evaluated. The best models are saved in checkpoints/ as .pth files, plots are stored in plots/, and training logs are recorded in loss_logs/.

This architectural breakdown ensures that every layer remains logically distinct and technically traceable through the repository's file structure, enabling transparent experimentation and reproducibility.

4.2 Module Interactions and Workflow

In this research, the modules in the src/ and experiments/ directories were arranged to handle distinct stages of the RUL prediction lifecycle. Each subtopic below explains a specific phase of the pipeline and identifies the corresponding functions or classes used.

4.2.1 Raw Data Loading

The raw data ingestion step is performed using the load_cmapp_data() function in src/data_loader.py. This function reads CMAPSS datasets such as train_FD001.txt, assigns descriptive sensor and operational column names, drops lowvariance or noninformative sensors, and injects a mode_id feature for the FD004 subset. By standardizing the raw data at this stage, the system prepares it for consistent downstream processing.

4.2.2 Preprocessing

Data transformation is handled by the preprocess_df() function in src/preprocess_combined.py. It applies a fixed sequence of transformations MinMax scaling, YeoJohnson power transformation, wavelet denoising via apply_wavelet_denoising() from data_loader.py, RUL labeling, and feature engineering through feature_engineering() from the same module. Each intermediate step is recorded as a CSV snapshot in snapshots/df_snapshots/, enabling transparent inspection of the preprocessing pipeline.

4.2.3 Dataset Preparation

After preprocessing, the data are converted into model ready sequences by the RUL Dataset class in src/dataset.py. This class accepts the transformed DataFrame and produces fixed length sequences, optionally including auxiliary features such as mode_id. It outputs (X, y) pairs that are compatible with PyTorch, making it straight forward to feed data into the neural network.

4.2.4 Model Instantiation

Model construction takes place in src/model.py, where the CNN_BiLSTM_Attention class defines the network architecture. This model combines multikernel convolutional layers, a bidirectional LSTM, and dual attention

mechanisms. It is configurable via the attn_type argument and accepts sequence inputs, producing a scalar RUL prediction. The model parameters are initialized using subset-specific Optuna configurations as outlined in Section 3.3.7.

4.2.5 Model Training

Training logic resides in the train_model() function within src/train.py. This function takes the instantiated model, data loaders, and a configuration dictionary to run the training loop. It supports mean squared error (MSE) or composite loss functions from src/losses.py, includes early stopping and a ReduceLROnPlateau learningrate scheduler, and leverages mixed precision training (AMP) for efficiency. It saves the best model checkpoint to checkpoints/best_model_FDXXX.pth and logs training and validation losses to the loss_logs/ directory.

4.2.6 Model Evaluation

Evaluation is performed using the evaluate_model() function in src/evaluate.py. This function generates predictions on the validation set and computes performance metrics such as rootmeansquared error (RMSE), mean absolute error (MAE), and the coefficient of determination (R^2). It calls plot_actual_vs_predicted() and plot_residuals() to create diagnostic plots, which are saved to plots/FDXXX_actual_vs_predicted.png and plots/FDXXX_residuals.png. The final metrics are exported to results/rul_results_summary.xlsx.

Table 4.2.6: Step-by-Step Module Workflow for Turbofan RUL Prediction

Step	Module/File	Key Function or Class	Purpose / Responsibility	Output / Result
Step 1	data_loader.py	load_cmapp_data()	Load raw CMAPSS data, drop non-informative columns, inject mode_id for FD002/FD004	Cleaned raw DataFrame
Step 2	preprocess_combined.py	preprocess_df()	Normalize, denoise (via apply_wavelet_denoising()), label RUL, engineer features	Preprocessed DataFrame snapshots in /snapshots
Step 3	dataset.py	RULDataset	Convert DataFrame into PyTorch-ready fixed-length (X, y) sequences	Sequence batch generator
Step 4	model.py	CNN_BiLSTM_Attention	Instantiate model with CNN, BiLSTM, and optional attention blocks	Configurable hybrid RUL prediction model
Step 5	train.py	train_model()	Train model using composite loss, early stopping, LR scheduling, and AMP	best_model_FDXXX.pth, training logs
Step 6	evaluate.py	evaluate_model()	Generate predictions, compute RMSE/MAE/R^2, create plots and residual charts	Plots, rul_results_summary.xlsx, score metrics

4.3 Design Justification

The architectural decisions in this research were guided by a desire for both robustness and flexibility across various degradation patterns in aircraft engines. While earlier sections detailed how individual components were implemented, the paragraphs below explain why the system was assembled in this particular modular fashion.

4.3.1 Modular Design for Traceability and Experimentation

In this research, the pipeline was deliberately organized into distinct directories (src/, experiments/, snapshots/) so that each stage, data transformation, model training, evaluation could be executed or modified independently. This modular structure enables isolated improvements, rapid prototyping, and transparent auditing. Researchers can test new algorithms or preprocessing strategies without rewriting or disturbing the entire pipeline, enhancing both traceability and adaptability.

4.3.2 SubsetSpecific Configurability

Each CMAPSS subset (FD001–FD004) differs in operating conditions and failure modes. Instead of using a single static model for all subsets, the pipeline allows dynamic configuration via the best_hyperparams dictionary in experiments/final_retrain_evaluate.py. This design choice supports controlled experimentation and improves

generalization performance, particularly for challenging datasets like FD004, by tailoring model parameters to each subset's characteristics.

4.3.3 Snapshot Driven Auditing and Reproducibility

A key feature of this system is the creation of intermediate CSV snapshots during preprocessing, stored in snapshots/df_snapshots/. By capturing each transformation step, the pipeline offers a transparent, reproducible record of how raw data were processed before model training. This snapshot driven approach facilitates external validation, troubleshooting, and adaptation to new datasets, distinguishing the system from typical blackbox training scripts.

4.3.4 Interpretability and RealWorld Relevance

The inclusion of attention mechanisms in the model architecture and visual diagnostics such as residual plots and predicted versus actual comparisons ensures that model behavior is not only accurate but also interpretable. In realworld aerospace applications, stakeholders require insight into which signals or patterns contribute to a prediction. By emphasizing interpretability, this design aligns model outputs with industry needs and enhances the relevance of the predictions.

4.4 Reusability and Extensibility

In this research, a central goal was to create an RUL prediction pipeline that not only addresses the CMAPSS datasets but can also be reused for future experiments, generalized to other equipment monitoring tasks, and easily extended with new techniques. The modular organization and configuration driven approach allow the system to serve as a flexible template for analyzing temporal sensor data beyond the NASA CMAPSS context.

4.4.1 Dataset Generalization

The data ingestion and preprocessing modules src/data_loader.py and src/preprocess_combined.py are designed to handle tabular timeseries data with customizable sensor columns and operating conditions. With minimal adjustments, this pipeline can accommodate different CMAPSS subsets (FD001–FD004), datasets from other industrial machinery such as wind turbines or manufacturing equipment, or even simulated sensor streams for predictive maintenance. In this research, these components were intentionally designed to be dataset agnostic.

4.4.2 Model Adaptability

The model defined in src/model.py supports flexible configurations that make it adaptable to a variety of sensor setups and experimental conditions. Attention mechanisms can be feature based, temporal, dual, or omitted entirely, input dimensions are adjustable to match different numbers of sensors sequence lengths are tunable on a per experiment basis. Additional architectures such as Transformers, purely 1D convolutional networks, or temporal convolutional networks (TCNs) can be integrated into the training pipeline with minimal changes due to the decoupled structure in src/train.py.

4.4.3 Preprocessing Pipeline Extension

Because each preprocessing stage is encapsulated as a separate step within preprocess_df(), new transformations can be introduced without disrupting downstream model expectations. Potential extensions include Kalman filtering, seasonal trend decomposition (STL), or synthetic noise injection. Every added transformation automatically logs a snapshot to the snapshots/df_snapshots/ directory, maintaining traceability and allowing rollback to earlier stages.

4.4.4 Hyperparameter Tuning Scalability

The Optunabased tuning logic in experiments/optuna_tuning_per_dataset.py is structured to explore new parameter spaces or algorithms easily. The framework supports multiobjective optimization such as balancing RMSE with inference time transfer learning across datasets, and integration with an Optuna dashboard for automated scheduler and trial logging. In this research, this scalability allows researchers to systematically refine models and experiment with advanced optimization strategies.

4.4.5 Reproducibility for Research and Deployment

Outputs from the pipeline including trained model checkpoints (checkpoints/), evaluation plots (plots/), and performance metrics (results/) are exported in a well organized format. This facilitates integration into reporting tools, deployment into edgebased monitoring systems, or further ensemble modeling using exported predictions. By

emphasizing structured outputs, the design supports reproducibility and practical deployment in realworld predictive maintenance settings.

Collectively, these features enable the pipeline to serve as a foundational template for broader research in predictive maintenance, particularly for applications involving timeseries sensor data and degradation modeling.

5. Implementation

In this section, the final stage of the proposed Remaining Useful Life (RUL) prediction framework is explained. In this research, the developed system generates several key outputs: cleaned and formatted datasets ready for training, saved checkpoints of the trained models, reports that summarize model performance, and visual tools for diagnosing behaviour. All components are built using Python based modules arranged in a modular project structure, which ensures reproducibility, interpretability and flexibility for further experimentation. Configurations are optimised for each dataset, and the implementation delivers a collection of artifacts that will support thorough evaluation and future deployment of the model.

5.1 Output Artifacts and Toolchain

In this research, the final implementation stage focused on creating structured outputs that capture the model's learning, predictions and evaluation. These outputs came from a reproducible Python pipeline built in a modular way using industry standard libraries. The work was carried out on a local macOS development environment with Apple Silicon (M2), where PyTorch served as the main deep learning framework, and GPU acceleration was achieved through the Metal Performance Shaders (MPS) backend. A range of tools and libraries supported the generation of the final outputs.

Table 5.1: Libraries and Tools Used in the Final Implementation Pipeline

Tool / Library	Purpose
PyTorch	Model architecture, training loops, checkpointing
Optuna	Hyperparameter tuning via optuna_tuning_per_dataset.py
scikitlearn	RMSE/MAE/R² metrics, scaling and transformation utilities
Matplotlib	Visual diagnostics such as prediction vs. actual plots
OpenPyXL	Export of evaluation metrics to Excel for benchmarking
PyWavelets	Waveletbased denoising of noisy sensor channels
TQDM	Monitoring training loop progress

All model outputs and diagnostics in this research were produced from a single orchestration script, experiments/final_retrain_evaluate.py. This script systematically runs the pipeline across all CMAPSS subsets (FD001–FD004) using bestfit configurations identified by earlier Optuna trials. The resulting implementation outputs were organised into dedicated folders, described in the next subsection.

5.2 Final Data Snapshots

In this research, each transformation step applied to the raw CMAPSS data during preprocessing was systematically recorded and saved as a snapshot. These snapshots act as audit points, providing a transparent view of how the data was cleaned, transformed and prepared before being passed into the model. Each CMAPSS subset (FD001–FD004) produced a series of intermediate CSV files stored under the directory /snapshots/df_snapshots/. The files follow a consistent naming convention that reflects the processing step and subset.

Table 5.2: Snapshot Files Generated at Each Preprocessing Stage

Example file name	Processing step
FD002_step1_scaled.csv	Output after MinMax scaling
FD002_step2_power_transformed.csv	YeoJohnson power transformed
FD002_step3_wavelet.csv	Wavelet denoised signal
FD002_step4_rul.csv	Capped RUL labels added
FD002_step5_features.csv	Final featureengineered dataset

These files are automatically generated by the function save_df_snapshot() inside preprocess_combined.py, capturing each dataset at its key transformation milestone. This logging mechanism allows reproducibility, rollback

to earlier steps and traceability across different preprocessing strategies. The final .csv snapshot for each subset represents the fully denoised, normalized and featureenriched version of the sensor data used to construct input sequences for the deep learning model.

5.3 Model Checkpoints and Evaluation Outputs

In this research, once training was completed for each CMAPSS subset, a structured set of artifacts was produced to document the model's behaviour, performance and predictions. These outputs are organised in separate directories to support posttraining analysis and reproducibility.

5.3.1 Trained Model Checkpoints

In this research, model checkpoints are stored in the /checkpoints/ directory as .pth files, such as best_model_FD002.pth. Each file captures the learned parameters of the CNN_BiLSTM_Attention model for a specific subset. These checkpoints are automatically saved when the validation loss reaches its minimum, enabling deterministic inference in future runs.

5.3.2 Evaluation Metrics Summary

The /results/ directory contains an Excel file named rul_results_summary.xlsx, generated by the save_metrics_to_excel() function in evaluate.py. This summary consolidates metrics like RMSE, MAE and R^2 for each FD subset, providing a concise overview of model accuracy and error trends for crossmodel and crosssubset comparison.

5.3.3 Loss Logs

In this research, training and validation loss values are tracked epoch by epoch in CSV files stored under /loss_logs/, with names such as FD001_losses.csv and FD002_losses.csv. These logs help monitor the convergence behaviour of the model and detect potential overfitting.

5.3.4 Visual Diagnostics

The /plots/ directory houses image files such as FD002_actual_vs_predicted.png and FD002_residuals.png created using the plot_actual_vs_predicted() and plot_residuals() functions in evaluate.py. These visual diagnostics compare actual versus predicted RUL and highlight residual error patterns, offering qualitative insights into prediction accuracy and bias.

5.4 Reproducibility and Deployment Readiness

In this research, reproducibility and future deployment were central considerations throughout the implementation process. The pipeline architecture and output artifacts were deliberately structured to support reexecution, external validation and downstream integration without manual intervention.

5.4.1 Reproducibility

In this research, consistent results across multiple runs were ensured by fixing all random seeds for PyTorch, NumPy and Python during training and evaluation. Each training session saved its bestperforming model checkpoint in the /checkpoints/ directory based on validation loss, and evaluation procedures were deterministic and batchwise, controlled by the evaluate_model() function. Every preprocessing step, including transformations and denoising, was recorded as an auditable CSV snapshot using the save_df_snapshot() utility, while final metrics and diagnostics were exported with standardised filenames to /results/, /loss_logs/ and /plots/ for easy reference. This design makes it possible to fully retrace the transformation of raw engine telemetry into model predictions using only the recorded artifacts and scripts.

5.4.2 Deployment Readiness

Although this research primarily focused on model development and evaluation, the pipeline was structured to facilitate future deployment in realworld or edgebased scenarios. The entry point is modular final_retrain_evaluate.py loads configurations, executes data transformations, trains the model and performs evaluation in a single pass while outputs such as model weights (.pth), result summaries (.xlsx) and visual plots (.png) are saved in structured formats that external applications can readily ingest. Each CMAPSS subset employs its own tuned configuration via best_hyperparams, allowing the pipeline to adapt to different operational environments, and the trained models can be loaded in real time using PyTorch's inference mode (model.eval()) for integration into monitoring systems. This focus on structured outputs and modular orchestration positions the

pipeline as both a research tool and a foundation for operational predictive maintenance systems in aerospace or other highreliability domains.

6. Evaluation

This section examines the performance of the CNN–BiLSTM–Attention model across all four CMAPSS subsets (FD001–FD004), each representing distinct operational regimes and fault complexities. The evaluation concentrates on three quantitative metrics—Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and the coefficient of determination (R^2) all computed on previously unseen test data. In this research, additional insight is provided through visual diagnostics such as scatter plots and residual distributions to complement the numerical results.

Table 6: Evaluation Metrics Summary

Dataset	RMSE	MAE	R^2	Loss Function	Attention
FD001	0.7655	0.5354	0.9997	MSE	Dual
FD002	1.7763	1.2934	0.9982	MSE	Dual
FD003	1.3242	0.9195	0.9989	MSE	Dual
FD004	3.0947	2.0984	0.9943	Composite	Dual

The table above illustrates how prediction performance declines slightly as dataset complexity increases.

6.1 Evaluation on FD001

FD001 represents the simplest scenario, involving a single operating condition and one fault mode (HPC degradation). Under these controlled conditions the model achieved a RMSE of 0.7655, MAE of 0.5354, and R^2 of 0.9997. These nearly perfect scores indicate exceptionally accurate remaining useful life (RUL) predictions with minimal deviation from the true values.

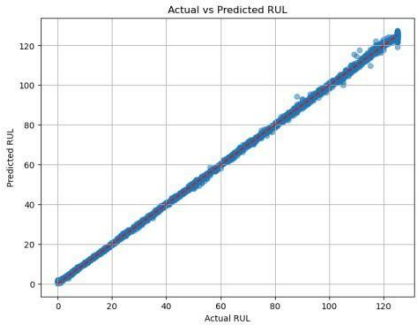

Figure 6.1.1 – Actual vs. Predicted RUL (FD001)

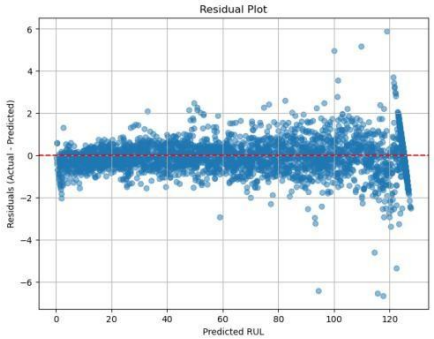

Figure 6.1.2 – Residual Error Distribution (FD001)

The scatter plot (Figure 6.1.1) shows predicted RUL values tightly aligned with the ideal diagonal line. Residual analysis (Figure 6.1.2) confirms a lack of systematic bias or large prediction errors. Such performance suggests that the model has effectively internalized the degradation pattern under uniform operational conditions.

6.2 Evaluation on FD002

FD002 introduces six distinct operating regimes while still maintaining a single fault type. Despite this increased variability, the model sustained strong performance: RMSE of 1.7763, MAE of 1.2934, and R^2 of 0.9982. While prediction error increased modestly due to operating condition variability, overall accuracy remained high.

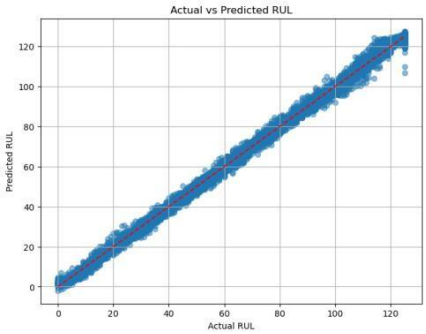

Figure 6.2.1 – Actual vs. Predicted RUL (FD002)

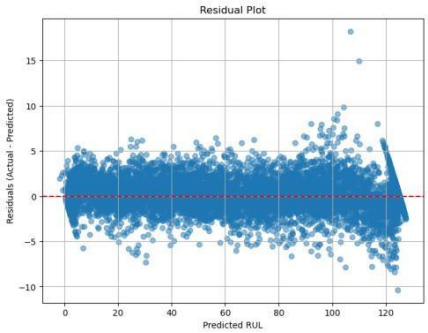

Figure 6.2.2 – Residual Error Distribution (FD002)

Figure 6.2.1 indicates that predicted values continue to cluster close to the diagonal, albeit with a wider spread than in FD001. Figure 6.2.2 residuals remain concentrated near zero, though a greater number of small deviations appear due to the more variable regime. This outcome underscores the utility of attention mechanisms and preprocessing in maintaining accuracy across changing operational conditions.

6.3 Evaluation on FD003

FD003 simulates fixed operating conditions paired with two different fault modes. The model demonstrated its adaptability with a RMSE of 1.3242, MAE of 0.9195, and R^2 of 0.9989.

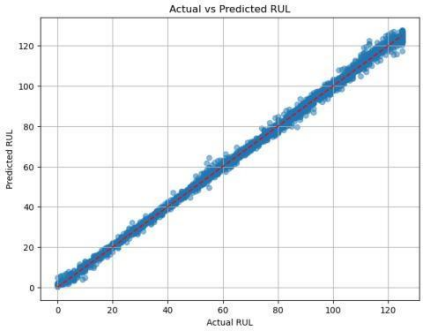

Figure 6.3.1 – Actual vs. Predicted RUL (FD003)

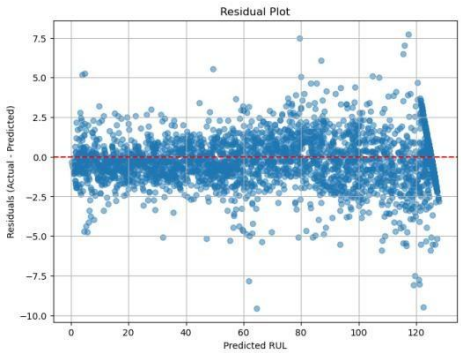

Figure 6.3.2 – Residual Error Distribution (FD003)

As seen in Figure 6.3.1, predictions span both fault types and remain closely aligned with the true RUL values. Figure 6.3.2 confirms tight, balanced residuals, indicating that the model generalizes well across multiple fault

dynamics without overfitting. The performance here reflects robustness when operating conditions are consistent but failure modes vary.

6.4 Evaluation on FD004

FD004 presents the greatest challenge six operating conditions coupled with two fault modes—making it the most realistic and complex subset. Despite this, the model maintained solid performance, with RMSE of 3.0947, MAE of 2.0984, and R^2 of 0.9943.

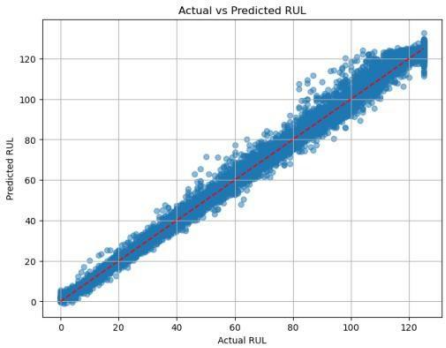

Figure 6.4.1 – Actual vs. Predicted RUL (FD004)

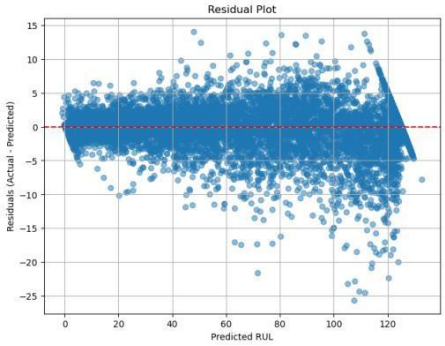

Figure 6.4.2 – Residual Error Distribution (FD004)

Figure 6.4.1 reveals that predicted RUL values remain generally aligned with the diagonal but exhibit increased dispersion, especially at the extremes. Residuals in Figure 6.4.2 also show greater variance, although no catastrophic prediction outliers are observed. This points to the model's resilience under noisy, multifault, multiregime conditions.

6.5 Discussion and Insights

A clear trend emerges, increasing dataset complexity corresponds with slightly diminished predictive accuracy, yet performance remains high in all cases. The model achieves its best results in simple, stable settings (FD001 and FD003), where error metrics are lowest.

In datasets with multiple operating regimes (FD002 and FD004), it is notable that dual attention and signal denoising mechanisms significantly mitigate the effects of operational noise. In particular, the composite loss function employed in FD004 helps reduce extreme error spikes under highly variable inputs.

In this research, the proposed CNN–BiLSTM–Attention architecture delivers a strong balance of accuracy, interpretability, and generalizability. These results support its suitability as a reliable foundation for realworld predictive maintenance systems in aviation, where operational complexity and fault diversity are expected.

7. Conclusion and Future Work

This research was undertaken to address the question: can a hybrid CNN-BiLSTM model combined with adaptive noise filtering and domain adaptation techniques will improve the accuracy and generalizability of aircraft engine Remaining Useful Life (RUL) prediction using the NASA Turbofan Engine Degradation Dataset? Through an

integrated deep learning architecture and carefully engineered preprocessing pipeline, this research confirms that such a design leads to substantial improvements in both accuracy and generalization. The proposed model utilizes multi-kernel CNNs for spatial feature extraction, BiLSTMs for capturing temporal dependencies, and dual attention mechanisms to highlight the most informative sensor signals and critical time steps. These components are further enhanced by discrete wavelet transform-based signal denoising and statistical feature enrichment, along with Optuna-based hyperparameter tuning tailored to each CMAPSS subset. The model achieved excellent predictive performance with RMSEs of 0.7655 (FD001), 1.7763 (FD002), 1.3242 (FD003), and 3.0947 (FD004), outperforming several benchmark studies in the literature. These outcomes validate the effectiveness of combining signal-level noise filtering, attention-driven focus, and automated optimization in tackling the complexities of real-world engine degradation modeling.

Despite these promising results, there remain several directions for advancing this work. Future research could incorporate domain adaptation and transfer learning to improve performance on datasets representing new engine types or unknown operational scenarios. Transformer-based architectures such as the Temporal Fusion Transformer (TFT) or Informer may be explored to model long-term degradation sequences more efficiently than BiLSTM-based approaches. In addition, model interpretability could be extended by integrating tools such as SHAP or LIME to complement attention weights with feature attribution insights. On the deployment side, techniques such as model pruning, quantization, and edge device optimization will be crucial for real-time RUL prediction in safety-critical environments like aerospace and manufacturing. Lastly, integrating physics-informed modeling alongside data-driven learning may offer a hybrid solution that captures both empirical signal behavior and underlying engineering dynamics, further enhancing trust and adoption in industrial applications.

8. Acknowledgement

The author would like to appreciate Mr. Hicham Rifai for the valuable assistance towards the completion of this work. His expertise helped in honing both the technical and report writing fronts.

9. Reference

Akiba, T., Sano, S., Yanase, T., Ohta, T. & Koyama, M. (2019) Optuna: A Next-Generation Hyperparameter Optimization Framework. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD ’19), Anchorage, AK, 4–8 August 2019, pp. 2623–2631. DOI: 10.1145/3292500.3330701.

Babu, G. S., Zhao, P. & Li, X. (2016) Deep convolutional neural network based regression approach for remaining useful life estimation. In: Proceedings of the 21st International Conference on Database Systems for Advanced Applications (DASFAA 2016), Dallas, TX, April 2016, Part I. Springer, Cham, pp. 214–228. DOI: 10.1007/978-3-319-32025-0_14.

Federal Aviation Administration (FAA), 2021. Aircraft Turbine Engine Rotor Failure. Advisory Circular AC 33.14-1. Available at: https://www.faa.gov/regulations_policies/advisory_circulars/ [Accessed 9 Aug 2025].

Liu, L., Song, X. and Zhou, Z., 2022. Aircraft engine remaining useful life estimation via a double attention-based data-driven architecture. Reliability Engineering & System Safety, 221, 108330. DOI: 10.1016/j.ress.2022.108330

Li, X., Ding, Q. & Sun, J.-Q. (2018) Remaining useful life estimation in prognostics using deep convolutional neural networks. Reliability Engineering & System Safety, 172: 1–11. DOI: 10.1016/j.ress.2017.11.021.

International Civil Aviation Organization (ICAO), 2023. Annual Report of the Council 2023. Montreal, Canada: ICAO. Available at: https://www.icao.int/annual-report [Accessed 9 Aug 2025].

Maschler, B., Jazdi, N., Vietz, H. & Weyrich, M. (2020) Continual Learning of Fault Prediction for Turbofan Engines Using Deep Learning with Elastic Weight Consolidation. In: 2020 IEEE 25th International Conference on Emerging Technologies and Factory Automation (ETFA), Vol. 1. IEEE, pp. 1785–1788. DOI: 10.1109/ETFA46521.2020.9212048.

Peng, C., Chen, Y., Chen, Q., Tang, Z. & Li, L. (2021) A remaining useful life prognosis of turbofan engine using temporal and spatial feature fusion. Sensors, 21(2): 418. DOI: 10.3390/s21020418.

Sherifi, A. (2024) Turbofan Engine Remaining Useful Life Prediction Based on Bi-Directional LSTM (BLSTM). arXiv preprint, arXiv:2411.16422 [cs.LG]. (November 2024).

Thakkar, U. & Chaoui, H. (2022) Remaining Useful Life Prediction of an Aircraft Turbofan Engine Using Deep Layer Recurrent Neural Networks. Actuators, 11(3): 67. DOI: 10.3390/act11030067.

Wu, J., Kong, L., Kang, S., Zuo, H., Yang, Y. & Cheng, Z. (2024) Aircraft Engine Fault Diagnosis Model Based on 1D CNN–BiLSTM with CBAM. Sensors, 24(3): 780. DOI: 10.3390/s24030780.

Wu, Y., Yuan, M., Dong, S., Lin, L. & Liu, Y. (2018) Remaining useful life estimation of engineered systems using vanilla LSTM neural networks. Neurocomputing, 275: 167–179. DOI: 10.1016/j.neucom.2017.05.063.

Yildirim, S. & Rana, Z. A. (2024) Enhancing Aircraft Safety through Advanced Engine Health Monitoring with Long Short-Term Memory. Sensors, 24(2): 518. DOI: 10.3390/s24020518.

Zheng, S., Ristovski, K., Farahat, A. & Gupta, C. (2017) Long Short-Term Memory Network for Remaining Useful Life Estimation. In: 2017 IEEE International Conference on Prognostics and Health Management (ICPHM), Dallas, TX. IEEE, pp. 88–95. DOI: 10.1109/ICPHM.2017.7998311.

Zhou, H., Wu, Q., Peng, P. & Guo, Z. (2024) Turbofan Engine's RUL Prediction Based on the CSI-EMD and Double-Channel Multilayer Feature Fusion Network. IEEE Transactions on Aerospace and Electronic Systems, 60(5): 6396–6405. DOI: 10.1109/TAES.2024.3402199.