Transformer-Based Time-to-Failure prediction for plane engine with NASA C-MAPSS multivariate timeseries

Lorenzo Suppa
s334021@studenti.polito.it
Politecnico di Torino
Turin, Italy

Vito Perrucci
s331543@studenti.polito.it
Politecnico di Torino
Turin, Italy

Tanguy Dugas du Villard
tanguy.dugasduvillard@studenti.polito.it
Politecnico di Torino
Turin, Italy

Abstract

Predicting a machine's remaining useful life is crucial to optimizing predictive maintenance efficiency, safety, and sustainability. Multiple approaches can estimate this duration. This report introduces a transformer-based approach that exploits the potential of its modified encoder part to capture the temporal evolution and patterns of multivariate time series. The performance of the model is evaluated against machine learning models and a recurrent neural network, demonstrating the solution's ability to achieve convincing accuracy on physical data extracted from a simulated aircraft engine.

Keywords

Time-to-failure prediction, Remaining Useful Life, Encoder regressor, Timeseries

1 Introduction

Maintenance has a crucial role in every industry. Especially in the plane industry, where an average of $2.9M is spent every year to maintain one single aircraft. 46% of this budget is spent for the engine only [2]. Opposed to Industry 4.0, where a failure in a machine is more likely to cost money and time to a company, a failure in the engines of a plane during its flight can lead to catastrophic scenarios. In both cases, avoiding these failures is mandatory and predictive maintenance is widely used [21], [16] [14]. Developing a predictive model to estimate the optimal time for maintenance can significantly optimize the maintenance schedule, reducing both component production and maintenance time. Additionally, it provides a continuous assessment of engine health, aligning with the SDG Decent work and economic growth, Industry, innovation and infrastructure and Responsible consumption and production. This paper presents an approach based on Encoder Regressors, leveraging the encoder component of a transformer, to predict the Remaining Useful Life (RUL) of aircraft engines using historical sensor data collected during flights. The study first provides an overview of the dataset used, followed by an exploration of relevant state-of-the-art solutions. Subsequently, the proposed method is introduced, and its performance is analyzed and discussed.

2 Data

The data we are working on come from [3], a Kaggle dataset initially uploaded by NASA and used for the prognostics challenge competition at the International Conference on Prognostics and Health Management (PHM08). The dataset, described by [11], is made of four subsets. Each subset is divided into a training and a testing part. On one hand, the training files contain multiple multivariate time series—referred to as *trajectories*—of data extracted by sensors on a (virtual) plane engine. Among them, pressures, temperatures, flows and rotation frequencies are measured cycle by cycle from the beginning of a flight, when the engine is in perfect health, until its failure. On the other hand, test files contain the same type of trajectories, but ending before the failure, therefore they're pieces of complete trajectories of various lengths. We are predicting the number of cycles between the end of the flight and the actual failure of the engine, the RUL. A more detailed list of the sensor and their location within the aircraft is presented in Fig. 1 and Tab. 1.

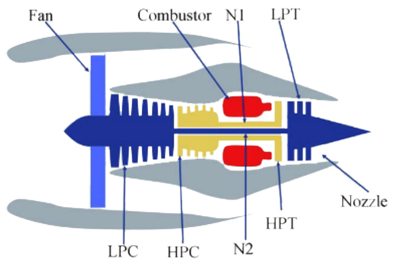
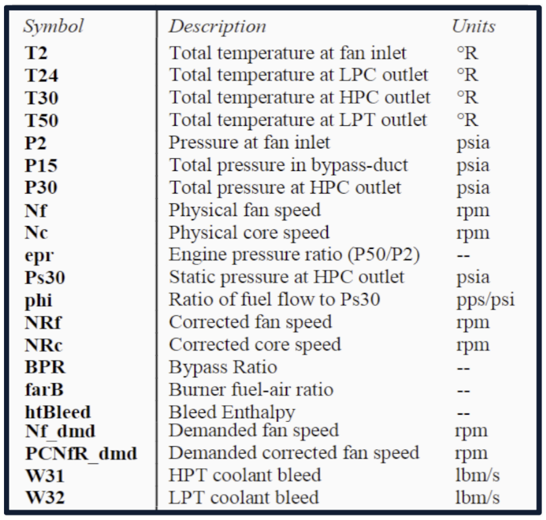


Figure 1: Aircraft's parts



Symbol	*Description*	*Units*
T2	Total temperature at fan inlet	°R
T24	Total temperature at LPC outlet	°R
T30	Total temperature at HPC outlet	°R
T50	Total temperature at LPT outlet	°R
P2	Pressure at fan inlet	psia
P15	Total pressure in bypass-duct	psia
P30	Total pressure at HPC outlet	psia
Nf	Physical fan speed	rpm
Nc	Physical core speed	rpm
epr	Engine pressure ratio (P50/P2)	--
Ps30	Static pressure at HPC outlet	psia
phi	Ratio of fuel flow to Ps30	pps/psi
NRf	Corrected fan speed	rpm
NRc	Corrected core speed	rpm
BPR	Bypass Ratio	--
farB	Burner fuel-air ratio	--
htBleed	Bleed Enthalpy	--
Nf_dmd	Demanded fan speed	rpm
PCNfR_dmd	Demanded corrected fan speed	rpm
W31	HPT coolant bleed	lbm/s
W32	LPT coolant bleed	lbm/s

Table 1: Features and sensor location

The four subsets contain data about different flight conditions. As an engine is a very complex machinery, failures can appear on different components. Also, a plane is able to fly in different conditions of speed, altitude and air temperature. While the first subset contains data related to a failure on the High-Pressure Compressor (HPC) and in one condition, the second subset presents six different flight conditions. The third subset relates to two failure types (the HPC and/or the fan) and one flight condition. Finally, the last one presents failures on both components and during six flight conditions.

3 Related work

Predictive maintenance of a machine is a topic increasingly studied for the last years, especially in the aircraft industry [16]. As some techniques involve some analytical and statistical approaches based on the physics of failures [5], machine learning and deep learning-based approaches are becoming the references in the domain. While the Autoregressive Integrated Moving Average (ARIMA) is one of the first approaches [6], a lot of applications are using basic machine learning techniques, such as Random Forests, Support Vector Machines, K-nearest neighbors or Linear and Polynomial regressions with their regularization variants, each of these approaches having pros and cons [16] and are the most used working with the NASA C-MAPSS dataset on Kaggle [3]. These algorithms are mainly used to directly estimate the RUL based on the last data sample, but they can also be used to estimate intermediate health metrics and indicators [4]. More recently, several approaches based on Neural Networks, such as Temporal Neural Networks [8], Long short-term Memories (LSTM) [7], or ensemble algorithms [15], provided better results, while other deep learning paradigms, such as reinforcement learning [13] and unsupervised learning [1] have been explored. Most recent techniques exploit transformers' capacities to catch the temporal evolution by developing transformer-based models [10], [18], or hybrid models [20], achieving even better results.

4 Method

In this section is presented the method used to process the data. The overall algorithm uses as input one trajectory representing the historical physical data of the engine, from perfect health to a given time, from which the algorithm predicts the remaining useful life.

4.1 Preprocessing

The data need first to be preprocessed. The multivariate times series includes 26 features. As some of them are highly correlated, a feature selection is done to improve the efficiency of the model. They also contain a complex noise that can't be statistically modeled [11] and should be removed. Finally, most machine learning algorithms, including the one we use, need inputs of a fixed shape. As the trajectory we use can have different lengths, we have to extract windows from them.

4.1.1 Feature selection. The first step of the preprocessing phase involves selecting the most relevant features. As shown in Fig. 2, some features exhibit high correlations, suggesting that a reduced subset is sufficient to develop an efficient model. The technique used to select this subset is **backward elimination**, as described in [9]. This iterative method removes features that contribute the least information, based on the p-value derived from ordinary least squares linear regressions. Specifically, the process begins with a table of features, a vector of target variables, and a threshold between 0 and 1. A linear regression is then computed on the features, and for each feature, the null hypothesis (that the corresponding coefficient is equal to 0) is defined and tested. The resulting p-values are compared to the predefined threshold, and the feature with the highest p-value is identified. If this p-value exceeds the threshold, the feature is removed, and the process repeats. Otherwise, the elimination process terminates, and the set of eliminated features is returned.

By applying this algorithm to the initial dataset with a threshold of 0.1 and jointly considering the correlations between features, we eliminated the following:

- **Nf**, due to its high correlation with Nc, NRf, and NRc, and its significantly high p-value in the backward elimination process.
- **farB**, as its p-value in the backward elimination process was higher than 0.1.
- **NRc**, given its strong correlation with NRf and Nc.
- **P15**, **P30**, and **epr**, as they are highly correlated with P2.
- **T24** and **T50**, since they are strongly correlated with T2 and T30.

Although the Mach Number feature has a high p-value in the backward elimination process, it represents one of the three flight conditions. Therefore, it was retained to ensure the model can generalize across various flight conditions.

4.1.2 Denoising. The multivariate time series we are using presents a lot of noise. This noise can be an issue for the performance of the algorithm as it will influence the training of the model. Several methods can be used to mitigate the impact of the noise on the prediction, it has been chosen to remove the noise thanks to a denoising algorithm. We tried several denoising algorithms: Exponential smoothing, Wiener, and Savitzky-Golay (Savgol). Given the similar performance (as shown in Fig. 3), we chose the Savgol [12] filter for its distinctive advantages:

- **Preservation of Signal Features:** The Savitzky-Golay filter utilizes local polynomial fitting, which facilitates effective smoothing while maintaining essential signal characteristics such as peaks, valleys, and inflection points. This attribute is particularly critical for sensor data, where the accurate retention of signal features is of paramount importance.
- **Flexibility and Adaptability:** The filter's performance is highly tunable through adjustments of the window size and polynomial order. This flexibility enables it to effectively manage non-stationary data and variable noise levels, conditions frequently encountered in sensor measurements.
- **Robustness Relative to Alternatives:** Unlike the Wiener filter, which depends on the assumptions of signal and noise stationarity and requires precise estimation of noise statistics, or exponential smoothing methods that are primarily designed for forecasting purposes, the Savitzky-Golay filter provides a robust and generally applicable solution for denoising sensor data without reliance on stringent noise model assumptions.

In summary, the Savitzky-Golay filter was chosen for its balanced performance, robust feature preservation, generalizability, and adaptability to the specific challenges inherent in sensor data

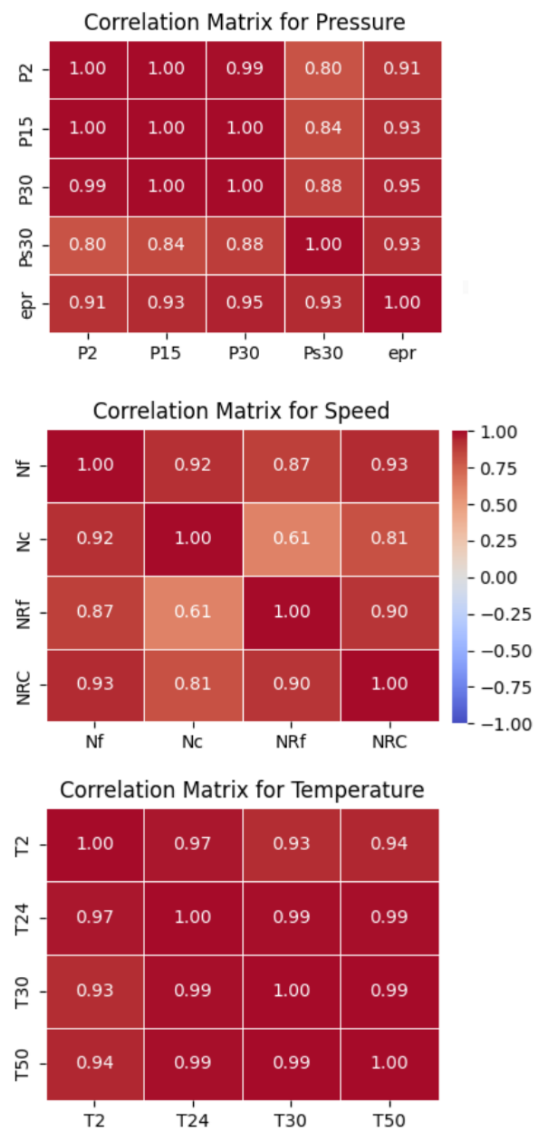


Figure 2: Views on the correlation matrix

processing. The effect of the denoising on a few features is shown on Fig. 4.

4.1.3 Windowing. The dataset contains trajectories of various lengths, as shown in Fig. 5. While Transformers architectures can process variable-length inputs due to their self-attention mechanism, our application benefits from segmenting the time series into fixed-size windows (See Sec. 4.2 for further details). This approach enables us to structure the training data by associating each window with a specific Remaining Useful Life (RUL) corresponding to the RUL of the last cycle of the window. By extracting fixed-sized segments from trajectories, we standardize the inputs for batch processing and simplify the alignment of cycles with their RUL labels during training. The windowing algorithm relies on two hyper-parameters:

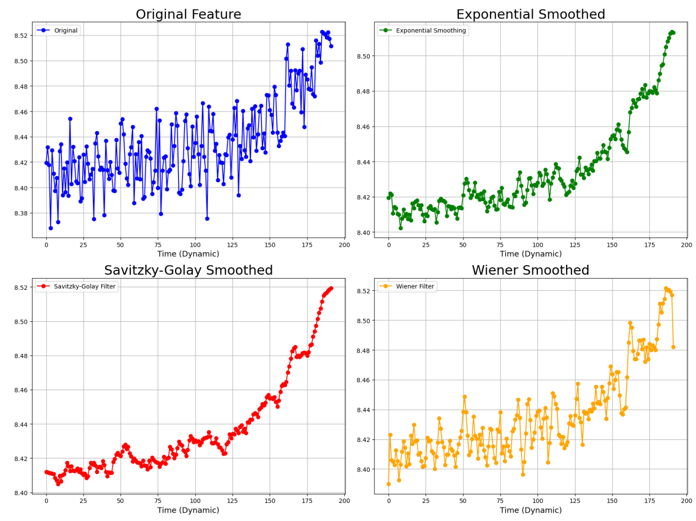


Figure 3: Denoising instance

window size and step size. We use a step size of one to maximize the number of windows extracted from each trajectory, thereby enriching the training dataset.

4.2 Proposed model

Our approach leverages the Encoder part of Transformer's architectures to address a regression task by first pre-training the model using a masked prediction objective and then fine-tuning it for end-to-end regression. This two-stage training process is designed to enable the model to learn robust feature representations from the sequential input data and subsequently adapt these features for precise regression predictions. A more detailed breakdown of our architecture and training strategy is provided in the following sections. During the detailed description of each layer of the model, we will explicitly provide the corresponding hyperparameter values, which were carefully selected through an extensive grid search performed on the first dataset, ensuring that our architecture is optimally configured for performance and robustness.

4.2.1 Pretraining with Masked Prediction Model. The initial phase of our training employs a masked prediction objective inspired by methods commonly used in natural language processing. The model, termed the `MaskedPredictionModel`, is designed to reconstruct the original input signal from a version where a random subset of the input values has been masked (i.e., set to zero). Each input value is masked with a 15% probability. This encourages the network to learn contextual representations that capture both short- and long-range dependencies in the data.

Input Embedding and Positional Encoding. The input data, which consists of multiple features over a sequence, is first passed through a linear embedding layer that projects it into a higher-dimensional space (of dimension d_{model}=**256**). Let the input embedding be denoted by

$$E \in \mathbb{R}^{T \times d_{\text{model}}},$$

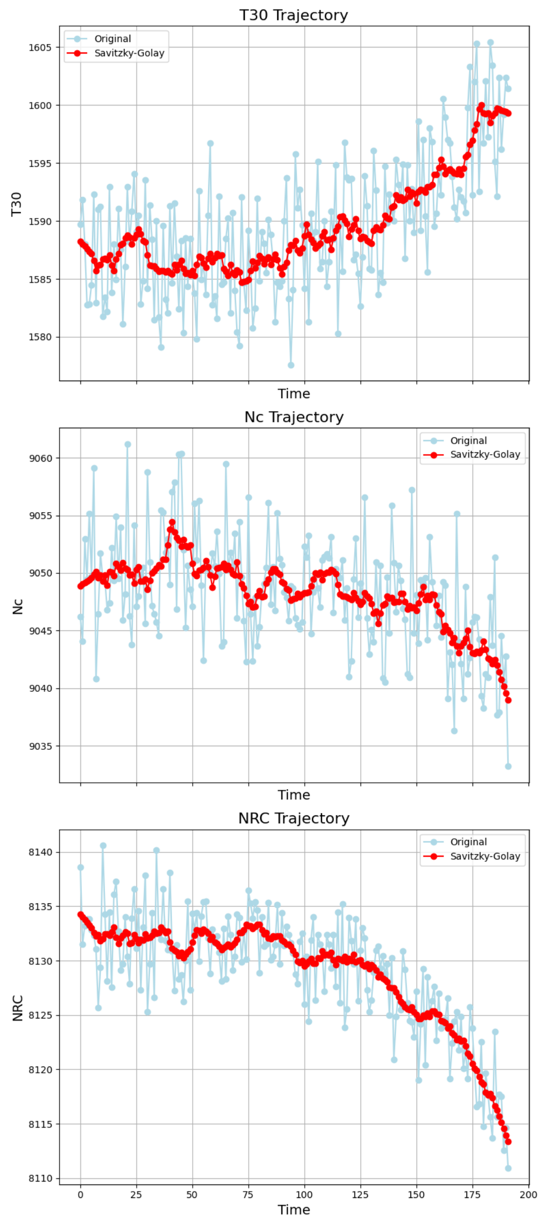


Figure 4: Three features before and after denoising

where T is the sequence length. To preserve the order of the sequence, we explore two distinct techniques for incorporating positional information:

(1) **Learnable Positional Encoding**
In our primary approach, we define a learnable positional encoding tensor

$$P \in \mathbb{R}^{T \times d_{\text{model}}},$$

which is initialized with random values (typically drawn from a normal distribution). During the forward pass, the positional encoding is added element-wise to the input embedding:

$$X = E + P.$$

This operation augments each element in the sequence with a unique, learnable positional bias, enabling the model to capture the order and relative positions of the tokens adaptively.

(2) **Fixed Sinusoidal Positional Encoding**
As an alternative, we compute a fixed sinusoidal positional encoding, inspired by the formulation in [17]. This encoding is defined by:

$$\text{PE}(pos, 2i) = \sin\left(\frac{pos}{10,000^{2i/d_{\text{model}}}}\right),$$

$$\text{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10,000^{2i/d_{\text{model}}}}\right),$$

where pos is the position index and i indexes the embedding dimensions.

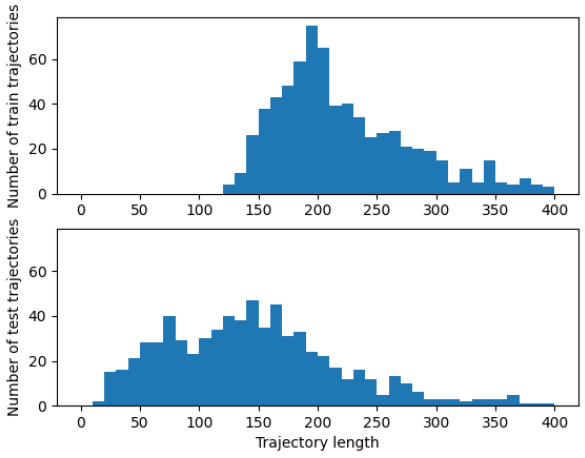


Figure 5: Distributions of the length of the trajectories in the train and test sets

In our experiments, we compare the model's performance using these two different positional encoding strategies. The learnable approach allows the model to adjust the positional information during training, while the fixed sinusoidal encoding offers a deterministic method to inject sequence order without additional parameters.

Custom Transformer Encoder Layers. At the core of our model lies a stack of **4** custom Transformer Encoder layers. Each layer comprises the following components:

- **Multi-head Self-Attention:** In our model, multi-head self-attention is computed following the formulation introduced by [17]. Given an input tensor

$$X \in \mathbb{R}^{T \times d_{\text{model}}},$$

where T is the sequence length, and d_{model} is the hidden dimension, the process unfolds as follows:

(1) **Linear Projections:** The input tensor X is linearly projected into queries, keys, and values:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V,$$

where $W^Q, W^K, W^V \in \mathbb{R}^{d_{\text{model}} \times d_h}$ are learnable weight matrices and d_h is the dimensionality of each head.

(2) **Scaled Dot-Product Attention:** For each head, the attention scores are computed by taking the dot product between queries and keys, scaling by the square root of the key dimension to mitigate the effect of large dot-product values, and applying a softmax to obtain the attention weights:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_h}}\right)V.$$

(3) **Multi-head Formulation:** The process above is executed in parallel for **16** heads. The outputs of each head are concatenated to form a single tensor:

$$\text{MultiHead}(X) = \text{Concat}\big(\text{head}_1, \ldots, \text{head}_H\big)W^O,$$

where H=**16** is the number of attention heads, each head$_i$ is computed as described above, and $W^O \in \mathbb{R}^{(H \cdot d_h) \times d_{\text{model}}}$ is a learnable weight matrix that projects the concatenated output back to the model dimension.

This multi-head self-attention mechanism enables the model to capture diverse inter-positional relationships in the input sequence by allowing it to attend to different parts of the sequence in parallel.

- **Residual Connections and Dropout:** After computing the self-attention output *SA(X)*, a residual connection is applied by adding the original input to the dropout-regularized self-attention output. This dropout helps in regularizing the model and prevents overfitting by randomly zeroing some of the elements of the self-attention network output during training with a **20%** dropout probability. Formally,

$$\tilde{X} = X + \text{Dropout}\big(\text{SA}(X)\big).$$

A similar procedure is applied after the feedforward network sub-layer, further explained.

- **Batch Normalization:** In our implementation, batch normalization is applied after each residual addition. For a tensor $\tilde{X} \in \mathbb{R}^{T \times d_{\text{model}}}$, the normalization process is applied along the feature dimensions (second axis):

$$\hat{X} = \text{BatchNorm}\big(\tilde{X}\big).$$

The choice of batch normalization over the conventional layer normalization (commonly used in NLP Transformers) is motivated by the presence of outliers in our data. By aggregating statistics over both the batch and sequence dimensions, batch normalization more robustly manages such anomalies, ensuring consistent output distributions across layers.

- **Feedforward Network:** Following the self-attention sublayer and its associated normalization, the feedforward network first projects the input into a **128**-dimensional space via a linear layer. After applying the ReLU activation, the subsequent linear layer projects the 128-dimensional representation back to **256** dimensions. This process can be mathematically expressed as:

$$\text{FFN}(\hat{X}) = \text{Linear}_2\Big(\text{ReLU}\big(\text{Linear}_1(\hat{X})\big)\Big).$$

Next, dropout is applied to the output of the feedforward network in a similar way as previously done after the self-attention output.

$$\text{DFFN}(\hat{X}) = \text{Dropout}\big(\text{FFN}(\hat{X})\big).$$

A residual connection is then established by adding the dropout-regularized feedforward output to the original input:

$$\underline{X} = X + \text{DFFN}(\hat{X}).$$

To stabilize the training, batch normalization is applied to $\underline{X}$ in the same way we explained before:

$$\widetilde{X} = \text{BatchNorm}\big(\underline{X}\big).$$

Output Layer for Reconstruction. After processing through the Transformer encoder layers, the model employs a final linear layer that maps the high-dimensional representations back to the original input dimension. During pre-training, the Mean Squared Error (MSE) loss is computed only over the positions that were masked in the input.

4.2.2 *Transfer Learning and Fine-Tuning for Regression.*

Following the pre-training phase, the learned weights are transferred to a regression model, referred to as the `EncoderRegressor`. This model largely mirrors the structure of the `MaskedPredictionModel` with the following modifications:

Regression Head. Instead of reconstructing the input, the `EncoderRegressor` aggregates the sequential information using a mean pooling operation across the sequence dimension. The pooled representation is then passed through a fully connected regression head, which comprises a two-layer neural network that outputs a single prediction value.

Weight Transfer. The pre-trained weights from the masked prediction model are loaded into the `EncoderRegressor`. This weight initialization enables the regression model to benefit from the robust feature representations learned during pre-training.

4.2.3 *Training Strategy.*

Pre-training Stage. The model receives input windows in batches of **128**. In the pre-training stage, the masked prediction model is trained for a limited number of epochs. A masking strategy is applied where approximately 15% of the input values are randomly replaced with zeros. The model is optimized using the Adam optimizer (with a learning rate of 1×10^{-4} and weight decay regularization), and the MSE loss is calculated solely on the masked positions.

Fine-Tuning Stage. Again, even in this phase, the model receives input windows in batch of **128**. After transferring the pre-trained weights, the `EncoderRegressor` is fine-tuned on the regression task. Training involves standard procedures with both training and evaluation loops. An early stopping mechanism based on the validation loss is employed to prevent overfitting. The loss function during fine-tuning is again the MSE loss, ensuring consistency with the pre-training objective.

Overall, this two-stage training strategy (pre-training with a masked prediction objective followed by regression fine-tuning) allows the model to capture complex temporal dependencies in the data and achieve accurate regression predictions.

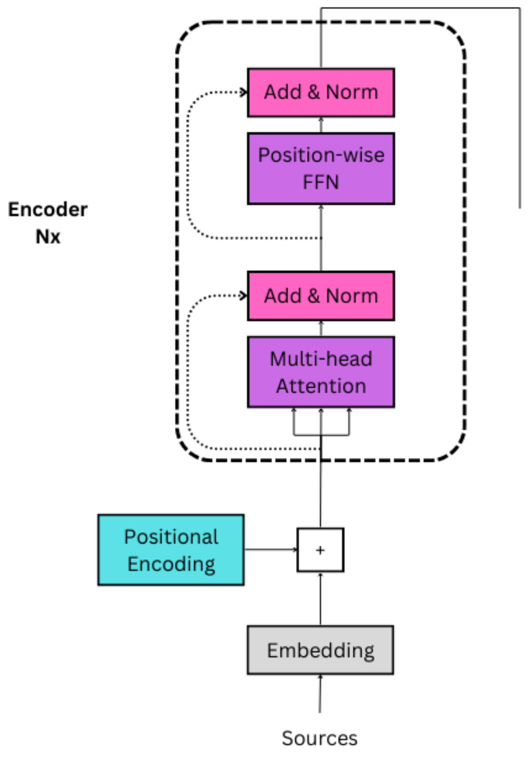


Figure 6: Illustration of the transformer-based model architecture.

4.3 Baseline models

While the encoder model previously described is the main solution, it has been compared to several simpler models, described below.

4.3.1 Linear Regressor. The simplest model for this task is a Linear Regression. A linear regressor estimates the best coefficient vector β by minimizing the prediction error E. The objective is to find :

$$\beta = \text{argmin } E(\beta) = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i^{\beta} - Y_i)^2$$

where the predicted value for entry i is given by:

$$\hat{Y}_i^{\beta} = \beta_0 + \sum_{j=1}^{m}\beta_j X_{i,j}$$

Here, Y_i is the ground truth for entry i, and $X_{i,j}$ represents the j-th feature of entry i. In our project, each input corresponds to a window, reshaped into a vector by concatenating its rows. This transformation allows the linear model to process sequential data as a standard regression problem, without explicitly considering temporal dependencies.

4.3.2 Random Forest Regressor. Random Forest Regressor is an ensemble learning method based on decision trees, designed to improve the predictive accuracy and robustness of individual trees. Each tree in the forest is trained on a random subset of the training data and considers a random subset of features at each split, reducing overfitting and enhancing generalization. Given a window X, each decision tree produces an independent prediction $\hat{y}_i$, and the final output is obtained by averaging the predictions from all N trees:

$$\hat{y} = \frac{1}{N}\sum_{i=1}^{N}\hat{y}_i$$

where N is the number of trees in the forest.

In our project, each input corresponds to a window, reshaped in the same way and for the same reasons already described in the previous subsection regarding Linear Regression. The predictions after the average correspond to the RUL estimated for specific windows, aggregating them we can construct the trajectory-level RUL predictions.

4.3.3 Long Short-Term Memory. Long Short-Term Memories [19] (LSTM) are a type of recurrent neural network (RNN) designed to capture temporal variations and dependencies in sequential data. Unlike standard RNNs, LSTMs include memory cells regulated by learnable gates (see Fig. 7): a **forget gate** to discard irrelevant information, an **input gate** to update the cell state, and an **output gate** to generate the hidden state. Given a window X, a first LSTM layer processes it and produces an output for each cycle in the window:

$$H^{(1)} = \text{LSTM}_1(X)$$

This output is then passed to a second LSTM layer, which condenses the information and returns a single vector:

$$h_T^{(2)} = \text{LSTM}_2(H^{(1)})$$

Finally, a dense layer predicts the Remaining Useful Life (RUL):

$$\hat{y} = W_{\text{dense}} h_T^{(2)} + b_{\text{dense}}$$

4.4 Postprocessing

As explained below, separating a trajectory in windows is used to create input of fixed shape. All these windows can be given to the model to output multiple RULs for the same trajectory. These RUL predictions can be aggregated to obtain the final prediction. This aggregation is done in two steps:

- First, the predictions need to be aligned. If a window finishing at time t on a trajectory of length T produces a RUL τ, it signifies that the failure would happen at time $t + \tau$, while we are trying to estimate τ such that the failure occurs at $T + \tau$. We simply add $T - t$ to all predictions.
- Then, several aggregation algorithms can be used. It has been chosen to compute an average of the prediction of the last 20 windows. If the prediction is above 40, it is kept,

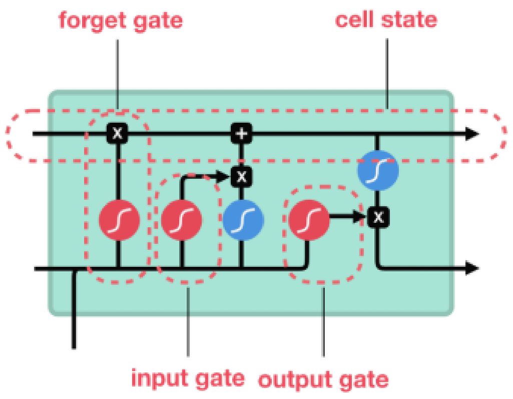


Figure 7: Schema of an LSTM layer

otherwise, an average over the last 9 windows is produced and returned. This algorithm is justified and accurately described in 6.4.

5 Results

In this section, we evaluate the performance of our `EncoderRegressor` at two distinct levels: first, at the individual window-level, and then after aggregating predictions across complete trajectories. It is essential to establish those performances, as they provide a critical reference point for understanding the impact of subsequent configuration adjustments. Assessing them, we will effectively isolate and analyze how variations in window size, positional encoding, and pretraining strategies contribute to its overall efficacy

5.1 Window-level performance

To evaluate the window-level performance of the previously defined `EncoderRegressor`, we measure the root mean square error (RMSE) of its predictions. While the RMSE can be computed over all windows, we further stratify the analysis by grouping windows according to their corresponding ground truth RUL. This approach allows us to assess the model's performance in relation to the specific RUL value associated with each window, with a particular focus on whether the window represents a low (imminent failure) or high RUL scenario.

As shown in Fig. 8, the model achieves notably lower RMSE values for windows corresponding to low RULs. This is a highly desirable outcome, since accurate predictions for low RULs trajectories are extremely important as they concern the trajectories the most at risk of failure.

5.2 Trajectory-level: performance of the algorithm

At this point, we proceed to analyze the forecasts at the trajectory level. To achieve this, we apply the previously described announced process, which will be strongly discussed in 6.4 later. This section is primarily intended to present the overall performance of the complete model, serving as a basis for future modifications and improvements.

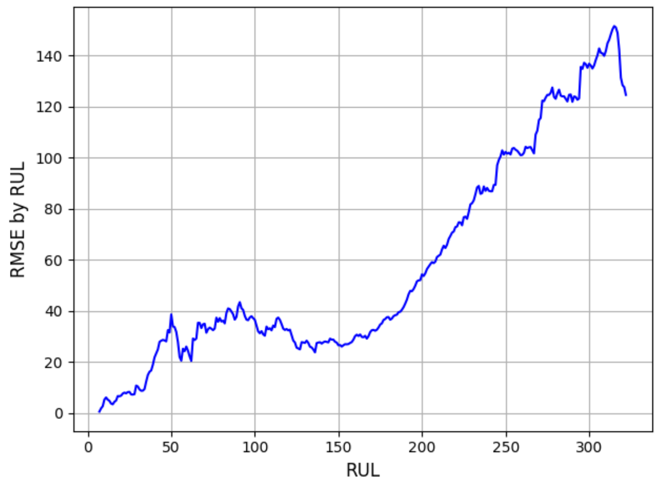


Figure 8: RMSE of the predictions vs actual RULs at window-level.

In this analysis, we report the results of two distinct experimental settings:

(1) Training and testing exclusively on the first subset.
(2) Training and testing on the entire dataset.

Fig. 9 illustrates the RMSE forecasts obtained after aggregating the predictions. To facilitate a more effective analysis, the RUL values on the x-axis have been grouped into bins of 20 units. This binning approach is motivated by our earlier window-level analysis, which underscored the importance of understanding how model performance varies with the RUL associated with each window.

The results demonstrate that the algorithm performs exceptionally well for short-term failures. As the RUL increases, the prediction error correspondingly rises, indicating a gradual decline in performance for higher RUL values. Specifically, the model achieves a global RMSE of 30.7 on the first subset. When trained on the full dataset, the algorithm maintains robust performance with a global RMSE of 35.2, with a slight decrease in accuracy.
This outcome highlights our model's capability to generalize effectively across varying flight settings and failure types.
Focusing on the performance for RUL values below 20, the results are particularly impressive. Specifically, the model trained and tested exclusively on the first subset achieved an RMSE of only 5.05, while the model trained and tested on the full dataset achieved an RMSE of 8.56.

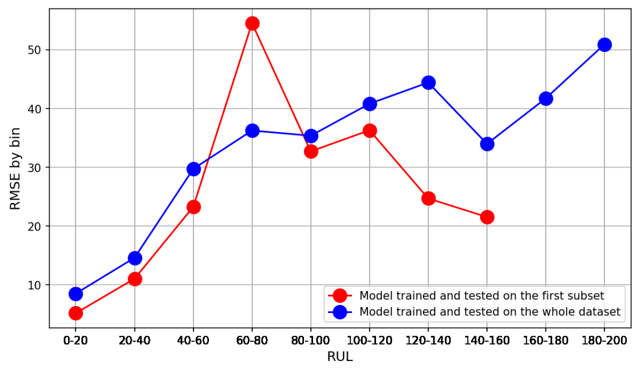


Figure 9: Final performance.

These results are particularly promising as the model demonstrates robust performance on components nearing failure, yielding accurate predictions even after the aggregation process where we have multiple windows for each trajectory, not just near-failure cycles. The evaluations presented above are performed on models with assumptions that follow extensive experiments conducted exclusively on the first dataset, during which various configurations (such as static versus learnable positional encodings) were systematically compared. The insights gained from these preliminary investigations were crucial in determining the optimal model settings and in refining the algorithm.

Moreover, the analysis provided here serves as a preliminary overview, in order to have an `EncoderRegressor` model that can be compared with the other additional ones. In the section 6 we will delve into a more detailed examination of the impact of various factors on performance, thereby offering further insights into potential avenues for future improvements.

5.3 Comparison against other models

As discussed in Sec. 4.3, three additional models were trained on the first subset and evaluated under the same preprocessing and postprocessing conditions. The performance comparison of these four models is illustrated in Fig. 10.

The results clearly indicate that the `EncoderRegressor` outperforms the alternative models, particularly in predicting short-term failures, which correspond to the most critical trajectories. The LSTM performs very well for medium-term failures but it exhibits significant errors for long-term failures and does not perform as well as the encoder in critical zones. This highlights the superior reliability of our model in the most crucial regions of prediction, confirming its suitability for this task. Despite the effectiveness of alternative methods, they do not achieve the same level of precision as our proposed model, further validating its selection for this specific application.

6 Discussions

As mentioned at the conclusion of Sec. 5.2, the choice of the `EncoderRegressor`'s settings requires thorough discussions. In this section, we provide a detailed analysis of the various design choices and their impact on model performance. Specifically, we

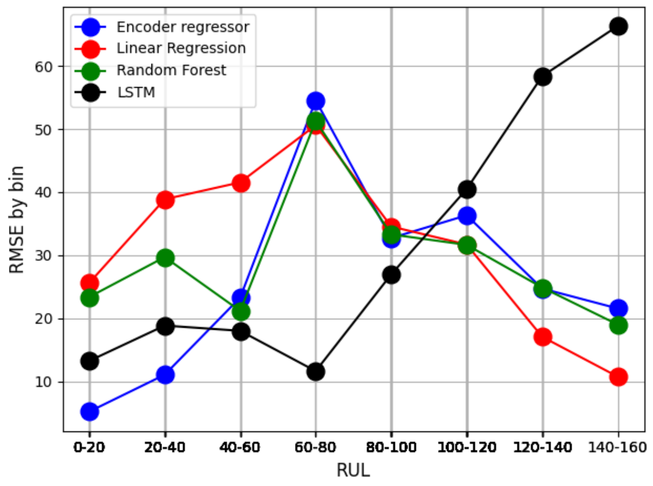


Figure 10: Comparison of the performance of the Encoder-Regressor against other models.

examine the effect of different window sizes, the role of positional encoding, the influence of pretraining, and the selection of the most effective aggregation strategy.

6.1 Window size

Let $T_{\min}$ denote the length of the shortest trajectory in the dataset. In our case, $T_{\min} = 19$, which constrains the window size w such that

$$2 \leq w \leq 19.$$

Although any w within this interval is admissible for training, our empirical analysis indicates that larger window sizes enable the `EncoderRegressor` to capture a more comprehensive temporal evolution, thereby enhancing performance.

To further investigate this phenomenon, we trained encoder regressors using window sizes of 30, 15, and 10.

The performance of each model was evaluated by computing the root mean squared error (RMSE) across different Remaining Useful Life (RUL) intervals, grouped in bins of 20 RUL units, using the first subset of trajectories.

As illustrated in Figure Fig. 11, larger windows yield lower RMSE values. Notably, while a window size of 30 produced the best performance, it exceeds $T_{\min}$ and is thus impractical for predicting the RUL for all trajectories.

The superior performance observed with longer windows is likely attributable not only to the extended temporal context they provide but also to the intrinsic benefits of the transformer-based architecture. Specifically, our Encoder Regressor appears to leverage longer input sequences more effectively, capturing richer temporal dynamics that contribute to more accurate RUL predictions.

In summary, although window sizes are bounded by the minimum trajectory length $T_{\min}$, our findings suggest that maximizing the window size (within these constraints) improves the predictive accuracy of our model. This enhancement is due to both the increased temporal information available and the improved performance of the transformer's encoder when processing longer sequences.

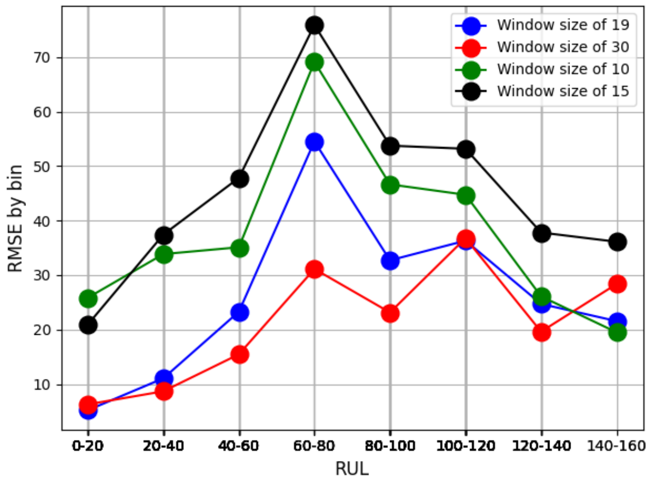


Figure 11: Performance of the model for different window sizes.

6.2 Learnable positional encoding and Static positional encoding

We investigated the impact of two distinct positional encoding approaches. Initially, our objective was to assess an alternative to the conventional, static positional encoding commonly employed in Natural Language Processing (NLP), anticipating that the model's behavior might differ under a learnable scheme. As the results presented in the accompanying figure Fig. 12 indicate, our hypothesis was well-founded. Specifically, the learnable positional encoding enabled the model to achieve markedly improved performance when predicting near-failure Remaining Useful Lives (RULs). For instance, in cases where RULs were less than 20, the learnable encoding outperformed the static approach by approximately 60%. In contrast, within the mid-term failure range, the static positional encoding demonstrated more robust performance, while in the late-term failure range, both encoding methods yielded comparable results. In conclusion, our findings suggest that learnable positional encoding is a promising strategy for applications involving transformer-based architectures beyond the traditional NLP context.

6.3 Pretraining

We evaluated the impact of pretraining, as detailed in the previous subsection 4.2.1, to determine its effectiveness in enhancing the model's performance. The results, as illustrated in the accompanying figure Fig. 13, reveal that pretraining contributes to significant performance improvements across the entire range of Remaining Useful Life (RUL) predictions. This enhancement is likely due to the pretraining process facilitating the model's ability to capture and utilize global temporal dependencies more effectively. Consequently, the model demonstrates a more robust and accurate

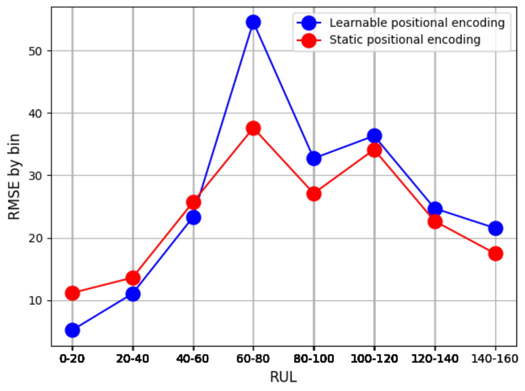


Figure 12: Comparison of the performance of the model with learnable and static positional encoding.

prediction capability, underscoring the value of incorporating pretraining in improving its overall generalization and performance.

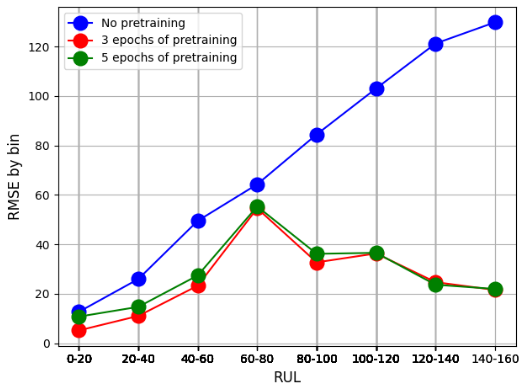


Figure 13: Comparison of the performance of the model with and without pretraining.

6.4 Aggregation algorithm

In Sec. 4.4, we already discussed the alignment required for predictions to be associated with the same trajectory. Once this alignment is done, we focus on the strategies for aggregating the predictions from the different windows of a trajectory. Several strategies can be adopted, all of which are applied to a subset of the last windows, whose number will be determined later. The strategies considered are as follows:

- **Unweighted average**: The average of all predictions is calculated without applying any weights.
- **Average weighted by cycle**: The predictions are linearly weighted based on the cycle number, with higher weights for later cycles.
- **Average weighted by $\sqrt{\textbf{cycle}}$**: The predictions are weighted by the square root of the cycle number, giving more importance to later cycles in a non-linear way.

- **Average weighted by** exp(**cycle**): Exponential weighting is applied, giving more importance to the few last cycles.
- **Average weighted by** log(**cycle**): Logarithmic weighting is applied, where later cycles are weighted more heavily, but the growth rate decreases.
- **Average weighted by cycle^2**: The cycle numbers are squared and used as weights, significantly emphasizing later cycles.

We applied the proposed aggregation algorithms to the predictions of the `EncoderRegressor` on the testing set of the first subset. In our analysis, we varied the number of windows used for aggregation and separated the results by short-term and long-term failure cases. As illustrated in Fig. 14, the optimal number of windows is dependent on the time remaining until failure; nonetheless, the unweighted average consistently delivers the best overall performance in both scenarios. Rather than selecting a single window count as a trade-off between short-term and long-term performance, we propose using two distinct window counts (one for short-term and another for long-term failure trajectories). This approach requires addressing two key points:

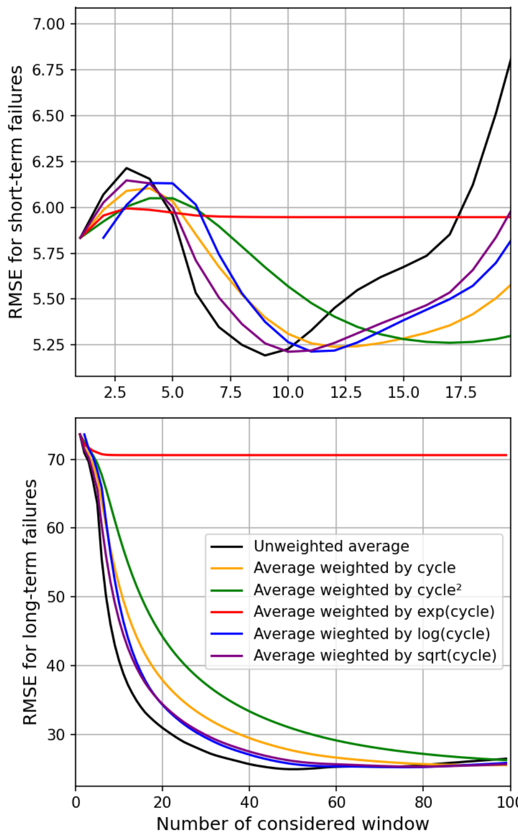


Figure 14: Comparison of aggregation methods.

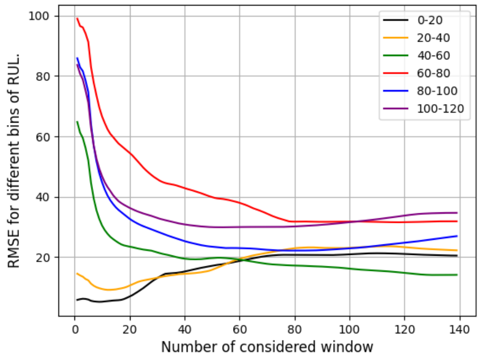


Figure 15: Performance of the aggregation algorithm for different bins of RULs.

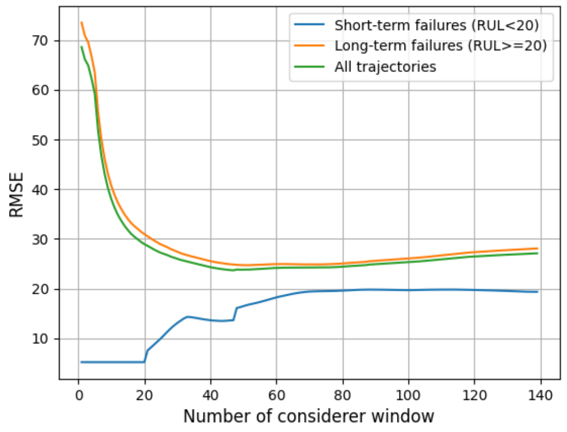


Figure 16: Performance of the algorithm based on the number of windows used for classifying a trajectory.

(1) **Defining Short-term vs. Long-term Failures:** We first establish a criterion for distinguishing between the two failure modes. To achieve this, we computed the RMSE within bins of 20 windows and plotted it as a function of the window count (see Fig. 15). The analysis indicates that:
   - For trajectories with a Remaining Useful Life (RUL) below 40, aggregation over the last **9** windows yields optimal performance.
   - For trajectories with an RUL above 40, aggregation over the last **80** windows is preferable.

   Although a more granular adjustment could be made for each bin, the limited resolution (bins of 20) does not support a robust per-bin classification. Therefore, we adopt these two representative values for short-term and long-term cases, respectively.

(2) **Trajectory Classification Method:** Next, we must classify a given trajectory as short-term or long-term. We achieve

this by evaluating the average prediction over the last 20 windows. The choice of 20 windows was based on an extensive evaluation of the algorithm's performance for various window counts (refer to Fig. 16). While an aggregation window of 45 would minimize the overall RMSE (approximately 25), it disproportionately increases the RMSE for short-term failures. In contrast, using 20 windows for classification ensures a lower RMSE for critical (short-term) trajectories while still maintaining satisfactory performance for trajectories with longer RUL.

In summary, by employing two distinct window counts (9 for short-term and 80 for long-term failures) and classifying trajectories using the average prediction over 20 windows, we achieve a more tailored aggregation strategy. This strategy effectively balances the performance trade-offs, yielding improved accuracy for both failure regimes.

7 Conclusion and Further Extension

Although the proposed approach yields encouraging results, for industrial applications, an extended version of the model should be trained on a more diverse dataset that accounts for various flight conditions and incorporates failures in additional engine components.

In many cases, we only have access to small segments of the full test trajectory. However, in a real-world setting, the model's predictive performance would improve if the trajectory used for RUL estimation were as long as possible.

Furthermore, since the current data are generated from simulations, fine-tuning the model with real-world data is highly recommended.

Another promising extension involves reformulating the task as a classification problem. In this scenario, different ranges of RUL could be defined and labeled (e.g., as "high-risk," "medium-risk," and "low-risk"). This classification approach would facilitate the development of a user-friendly application that is directly applicable to maintenance scheduling. However, due to limited domain-specific knowledge, it is necessary to consult with maintenance teams and management to accurately define these risk thresholds.

It is noteworthy that our model performs exceptionally well in predicting near-failure situations. We believe that incorporating these extensions would significantly enhance the model's performance, ultimately offering a high-performance solution for the company.

References

[1] Nagdev Amruthnath and Tarun Gupta. 2018. A research study on unsupervised machine learning algorithms for early fault detection in predictive maintenance. In *2018 5th International Conference on Industrial Engineering and Applications (ICIEA)*. 355–361. doi:10.1109/IEA.2018.8387124

[2] International Air Transport Association. 2022. Modern Airline Retailing: A Call for Change. https://www.iata.org/contentassets/bf8ca67c8bcd4358b3d004b0d6d0916f/fy2022-mcx-report_public.pdf

[3] Behrad3d. 2024. NASA CMAPS Dataset. https://www.kaggle.com/datasets/behrad3d/nasa-cmaps/data Accessed: 2025-01-31.

[4] Abel Diaz-Gonzalez, Austin Coursey, Marcos Quinones-Grueiro, Chetan S. Kulkarni, and Gautam Biswas. 2024. Data-Driven RUL Prediction Using Performance Metrics. In *35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024) (Open Access Series in Informatics (OASIcs), Vol. 125)*, Ingo Pill, Avraham Natan, and Franz Wotawa (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 21:1–21:15. doi:10.4230/OASIcs.DX.2024.21

[5] Shuai Fu and Nicolas P. Avdelidis. 2024. Aeronautics failure: a prognostic methodology based on the physics of failure and statistical approaches for predictive maintenance. In *NDE 4.0, Predictive Maintenance, Communication, and Energy Systems: The Digital Transformation of NDE II*, Christopher Niezrecki, Saman Farhangdoust, and Norbert G. Meyendorf (Eds.), Vol. 12952. International Society for Optics and Photonics, SPIE, 1295205. doi:10.1117/12.3018035

[6] Ameeth Kanawaday and Aditya Sane. 2017. Machine learning for predictive maintenance of industrial machines using IoT sensor data. In *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*. 87–90. doi:10.1109/ICSESS.2017.8342870

[7] Rodney Kizito, Phillip Scruggs, Xueping Li, Michael Devinney, Joseph Jansen, and Reid Kress. 2021. Long Short-Term Memory Networks for Facility Infrastructure Failure and Remaining Useful Life Prediction. *IEEE Access* 9 (2021), 67585–67594. doi:10.1109/ACCESS.2021.3077192

[8] Rongze Li, Zhengtian Chu, Wangkai Jin, Yaohua Wang, and Xiao Hu. 2021. Temporal Convolutional Network Based Regression Approach for Estimation of Remaining Useful Life. In *2021 IEEE International Conference on Prognostics and Health Management (ICPHM)*. 1–10. doi:10.1109/ICPHM51084.2021.9486528

[9] K.Z. Mao. 2004. Orthogonal forward selection and backward elimination algorithms for feature subset selection. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 34, 1 (2004), 629–634. doi:10.1109/TSMCB.2002.804363

[10] Oluwaseyi Ogunfowora and Homayoun Najjaran. 2023. A Transformer-based Framework For Multi-variate Time Series: A Remaining Useful Life Prediction Use Case. arXiv:2308.09884 [cs.LG] https://arxiv.org/abs/2308.09884

[11] Abhinav Saxena, Kai Goebel, Don Simon, and Neil Eklund. 2008. Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 International Conference on Prognostics and Health Management*. 1–9. doi:10.1109/PHM.2008.4711414

[12] Ronald W. Schafer. 2011. What Is a Savitzky-Golay Filter? [Lecture Notes]. *IEEE Signal Processing Magazine* 28, 4 (2011), 111–117. doi:10.1109/MSP.2011.941097

[13] Erotokritos Skordilis and Ramin Moghaddass. 2020. A deep reinforcement learning approach for real-time sensor-driven decision making and predictive analytics. *Computers Industrial Engineering* 147 (2020), 106600. doi:10.1016/j.cie.2020.106600

[14] Chandresh Soni, Gaurav Purohit, Anil Kumar Saini, and Chitra Gautam. 2025. *Aircraft Engine Condition Monitoring for Predictive Maintenance*. Springer Nature Singapore, Singapore, 435–455. doi:10.1007/978-981-97-8100-3_59

[15] Abhishek Srinivasan, Juan Carlos Andresen, and Anders Holst. 2023. Ensemble Neural Networks for Remaining Useful Life (RUL) Prediction. *PHM Society Asia-Pacific Conference* 4, 1 (Sept. 2023). doi:10.36001/phmap.2023.v4i1.3611

[16] Izaak Stanton, Kamran Munir, Ahsan Ikram, and Murad El-Bakry. 2023. Predictive maintenance analytics and implementation for aircraft: Challenges and opportunities. *Systems Engineering* 26, 2 (2023), 216–237. doi:10.1002/sys.21651 arXiv:https://incose.onlinelibrary.wiley.com/doi/pdf/10.1002/sys.21651

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL] https://arxiv.org/abs/1706.03762

[18] Peiwen Wang, Shaozhang Niu, Haoliang Cui, and Wen Zhang. 2024. GPT-based equipment remaining useful life prediction. In *Proceedings of the ACM Turing Award Celebration Conference - China 2024* (Changsha, China) *(ACM-TURC '24)*. Association for Computing Machinery, New York, NY, USA, 159–164. doi:10.1145/3674399.3674456

[19] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. 2019. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation* 31, 7 (07 2019), 1235–1270. doi:10.1162/neco_a_01199 arXiv:https://direct.mit.edu/neco/article-pdf/31/7/1235/1053200/neco_a_01199.pdf

[20] Hang Zeng, Hongmei Zhang, Jiansheng Guo, Bo Ren, Lijie Cui, and Jiangnan Wu. 2024. A novel hybrid STL-transformer-ARIMA architecture for aviation failure events prediction. *Reliability Engineering System Safety* 246 (2024), 110089. doi:10.1016/j.ress.2024.110089

[21] Tiago Zonta, Cristiano André da Costa, Rodrigo da Rosa Righi, Miromar José de Lima, Eduardo Silveira da Trindade, and Guann Pyng Li. 2020. Predictive maintenance in the Industry 4.0: A systematic literature review. *Computers Industrial Engineering* 150 (2020), 106889. doi:10.1016/j.cie.2020.106889