Comparing Feature and Trajectory-Based Remaining Useful Life Modeling of Electrical Resistance Heating Wires

Simon Mählkvist[1], Wilhelm Söderkvist Vermelin[2], Thomas Helander[3], and Konstantinos Kyprianidis[4]

[1,3] *Kanthal AB, Hallstahammar, Västmanland, 734 27, Sweden*
simonmkvst@gmail.com

[2] *RISE Research Institutes of Sweden, Mölndal, Västra Götaland, 431 53, Sweden*
wilhelm.soderkvist.vermelin@ri.se

[1,2,4] *Mälardalens University, Västerås, Västmanland, 721 23, Sweden*

ABSTRACT

Industrial heating significantly contributes to global greenhouse gas emissions, accounting for a substantial portion of annual emissions. The transition to fossil-free operations in the heating industry is closely linked to advancements in industrial electrical heating systems, especially those using resistance heating wires. In this context, Prognostics and Health Management is crucial for enhancing system reliability and sustainability through predictive maintenance strategies.

The integration of machine learning technologies into Prognostics and Health Management has significantly improved the precision and applicability of Remaining Useful Life modeling. This improvement enables more accurate predictions of component lifespans, optimizes maintenance schedules, and enhances operational efficiency in industrial heating applications. These developments are essential for reducing greenhouse gas emissions in the sector.

This paper serves as a guide for conducting Remaining Useful Life modeling for industrial batch processes. It evaluates and compares two methodologies: deep learning-based approaches using full time-series data, such as recurrent neural networks and their variants, and feature-engineering-based methods, including random forest regression and support vector machines. Our results show that the feature-oriented approach performs better overall in terms of predictive accuracy and computational efficiency. The study includes a detailed sensitivity analysis and hyperparameter estimation for each method, providing valuable insights into developing robust and transparent Prognostics and Health Management systems. These systems are crucial in supporting the heating industry's move towards more sustainable and emission-free operations.

The findings reveal that feature-oriented methods are both performant and robust, particularly excelling in handling outliers. The random forest regression model, in particular, demonstrated the highest performance on the test dataset according to the chosen evaluation metrics. Conversely, trajectory-oriented methods exhibited less bias across varying levels of degradation, a helpful characteristic for Prognostics and Health Management systems. While feature-oriented methods tend to systematically underestimate Remaining Useful Life at high true values and overestimate it at low actual values, this issue is less pronounced in trajectory-oriented models. Overall, these insights highlight the strengths and limitations of each approach, guiding the development of more effective and reliable predictive maintenance strategies.

1. INTRODUCTION

Industrial heating is a significant green house gas (GHG) emitter, contributing to approximately 22% of annual global emissions (Yoro & Daramola, 2020). Resistance heating wires play a crucial role in industrial electrical heating systems and offer a substantial opportunity for the heating industry's transition towards fossil-free operations.

In the prognostics and health management (PHM) field, the focus on maintaining industrial processes is continuously evolving. Specifically, within PHM, predictive maintenance strategies leverage data-driven modeling techniques, leading to the development of remaining useful life (RUL) prediction models.

The advent of machine learning (ML) technologies has sig-

nificantly enhanced the applicability and accuracy of RUL modeling. This advancement allows for more precise predictions of the lifespan of critical components, thereby optimizing maintenance schedules and improving operational efficiency in industrial heating applications.

Numerous challenges are associated with the development of modeling for industrial batch process data, and specifically, with the formulation of RUL models in this context. Within PHM, data quantity of full, end-of-line, trajectories are rare. The reason for this is that full trajectories involve destroying the component or product which can be costly. Simulation is not always feasible since first principles models may not exist for the degradation metrics, and developing said models is not an easy task.

The aim of this work is to bridge the modeling of RUL while evaluating different ML methods, focusing on improving prediction accuracy and robustness. As the original data is derived from time-series sensor trajectories data, two approaches will be employed to generalize this data.

Two approaches will be implemented and compared in this work in order to evaluate the sensitivity of said approaches. The first approach, referred to as the trajectory-oriented (TO) approach, will utilize the complete time-series data and the second approach, known as the feature-oriented (FO) approach, will align the data by extracting features from the time-series. There is a trade-off between the two approaches which makes the comparison interesting. The feature-oriented (FO) approach sacrifices the dynamics of the trajectories in order to be able to implement simpler methods, while the trajectory-oriented (TO) maintains the trajectory dynamics and have to rely on models of higher complexity.

1.1. System, sensors and data

This work approaches how to conduct RUL modeling of industrial batch process components and the data used provides resistance heating wires run-to-failure test data.

Accelerated degradation quality inspection test on resistance heating wires is used as a basis for the modeling and analysis conducted in this study. These test data contain full run-to-failure trajectories sampled for the wires in laboratory tests and consist of close to 1000 batches. The tests subject resistance heating wire to intense power cycling, accelerating degradation significantly.

Four sensors are used to monitor the test, made anonymous, named and denoted as such sensor 1 (S_1), sensor 2 (S_2), sensor 3 (S_3) and sensor 4 (S_4). In Figure 1, the four sensors are plotted for all batches. The data is min/max scaled from 0 to 1 where the batch with the longest duration is used as reference for 1. Further, the values of all four sensors are individually standardized, where the data are transformed to have an average value and a standard deviation of 0 and 1, respectively. Also, the vertical and horizontal axes correspond to the duration of the test and the sensor value, respectively. Further, the last point of each sample is emphasized by a black dot.

The distribution of the final points of the trajectories of each sensor are presented in Figure 2.

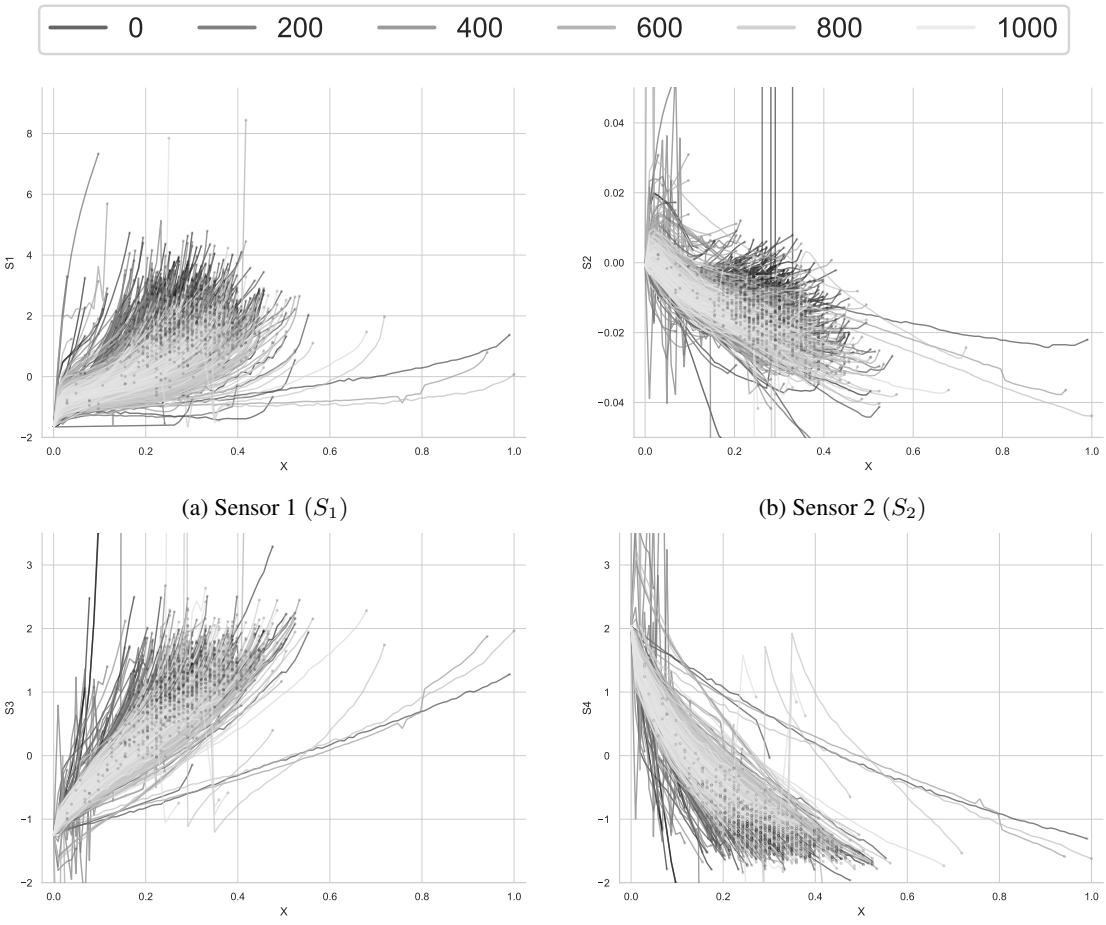
2. CONTRIBUTION

The main contribution of this study is twofold.

Firstly, this paper investigates RUL prediction on a novel dataset, gathered from experimental lab tests that simulate real usage conditions in a manufacturing process, although at harsher operating condition leading to an accelerated degradation. Such datasets, collected under controlled experimental settings rather than ongoing day-to-day operations, are valuable as they provide insight into the applicability of PHM techniques in a scenario that closely simulates *real-world* conditions. In particular, resistance heating wires are a class of assets that have received, to the authors' knowledge, little if any attention within PHM research. Secondly, this paper explores the different benefits and limitations of FO and TO-based RUL prediction. The FO approach relies on machine learning algorithms such as random forest regression (RFR) and support vector regression (SVR) that are proven to work well on tabular data. On the other hand, the TO approach relies on training on the full time-series data and are based on deep learning techniques. This work benchmarks and elaborates on the different methods, providing insights into trade-offs between the different approaches.

3. METHODOLOGY

RUL prediction can be achieved through various means and techniques. The main approaches are (Pecht & Kang, 2018; Galar, Goebel, Sandborn, & Kumar, 2021) 1) Data-driven, 2) Model based, 3) Hybrid methods. Data-driven methods revolves around gathering data from the asset of interest and develop models based on this data. E.g., machine learning models are a subset of such data-driven approaches. Model-based or physics-of-failure (PoF) models use first principles such as physics and mathematical modeling to develop prognostics models. Hybrid methods are a combination of both data-driven and model-based methods. Data-driven methods will be discussed in details below. Model based RUL prediction relies on an underlying model of the system at hand, such as a simulation, mathematical, or physical model. Such approaches are favorable in the case when there is a good physical understanding of the degradation process, and such, so-called PoF methods can be employed. Hybrid RUL prediction methods, aim to combine the two by using data to train data-driven models and combine the results with model-based techniques, such as PoF-approaches (Pecht & Kang,



(a) Sensor 1 (S_1)

(b) Sensor 2 (S_2)

(c) Sensor 3 (S_3)

(d) Sensor 4 (S_4)

Figure 1. Sensor Trajectories

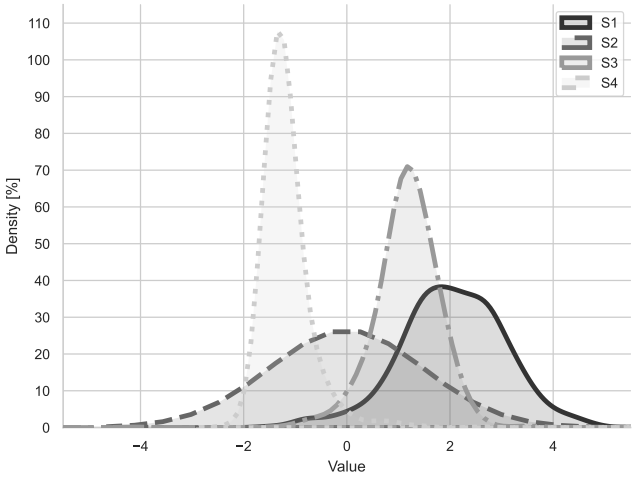


Figure 2. Sensor – Distribution

2018; Galar et al., 2021). This paper deals with data-driven approaches, based on machine learning methods.

An overview of the methodology, showing the successive steps of the shared framework, the individual approaches, as well as the shared concluding steps, can be seen in Figure 3. The details of the steps are found in the following sections, but in general, a common preprocessing is done followed by a random truncation of the trajectories. Then, the train/test split is designated which is followed by a split into the two separate approaches. The TO approach starts with an outlier removes, moves on to a train/validation split and concludes with a feature scaling. The FO, while similar to the former, starts with feature selection, continues to outlier removal, and finishes with removal of missing values.

3.1. Data Cleaning

As the data are provided and anonymized by the manufacturing company, most necessary data cleaning was performed prior to receiving the dataset. According to the company, the data have been normalized, extreme outliers have been removed, and other infeasible trajectories have been excluded

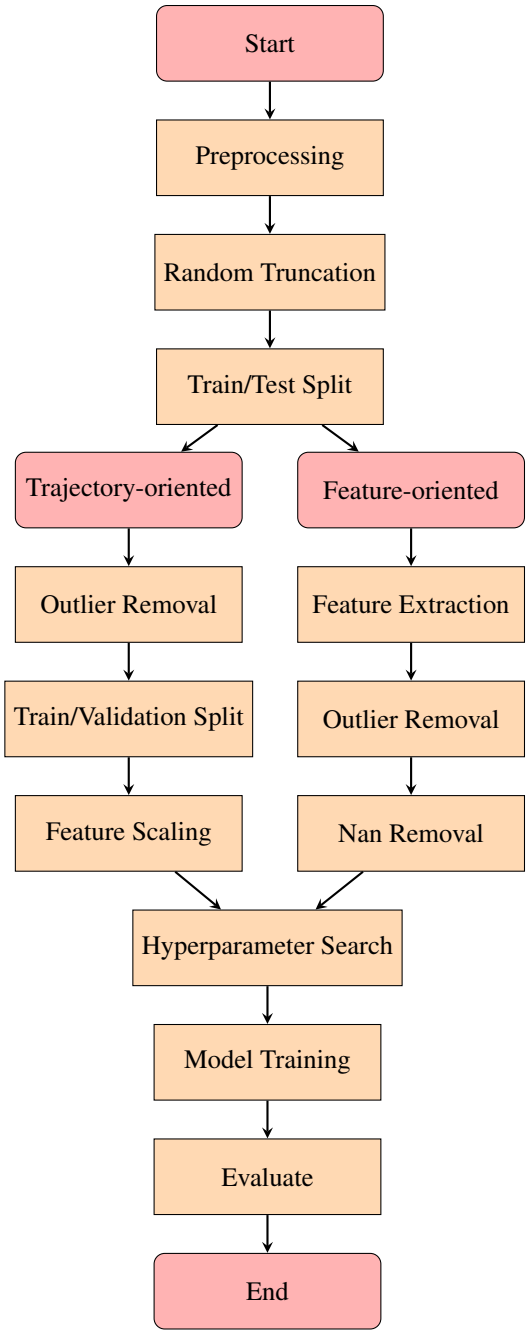


Figure 3. A flowchart representing the model development phases for both the FO and TO approaches.

from the data. Even still, it is important to perform some basic data pre-processing to prepare the data for machine learning modeling. Firstly, of course, the data are split into training, validation, and testing datasets. This is done so that the performance of the model can be assessed on the testing dataset. The validation dataset is used to monitor the training of the model, e.g., to stop training early in the case of overfitting. 20% of test identifiers (TIDs) of the full dataset is randomly allocated to the testing dataset. This means that out of the 695 total TIDs, 139 of them are assigned to the testing dataset. Then, out of the remaining 80%, another 80/20 split is performed, such that 64% of the original dataset is assigned to the training set, and 16% are assigned to the validation dataset. In Table 1 the partitions are shown.

Partition	%	# of TIDs
Full	100	695
Training	64	445
Validation	16	111
Testing	20	139

Table 1. Training, validation, and testing splits of the original dataset.

After partitioning the data into training, validation, and testing data, the data are scaled. There are several choices for scaling data but one of the common ways is so called min-max scaling where data are mapped onto to the interval $[a, b]$ where $a, b \in \mathbb{R}$ and $a < b$. Min-max scaling of a dataset $\mathbf{x}$ of size n is then defined in the following manner:

$$\tilde{\mathbf{x}}_i = \frac{\mathbf{x}_i - \min \mathbf{x}_{\text{train}}}{\max \mathbf{x}_{\text{train}} - \min \mathbf{x}_{\text{train}}} (b - a) + a, \tag{1}$$

for each $i = 1, \ldots, n$. Here, $\min \mathbf{x}_{\text{train}}$ and $\max \mathbf{x}_{\text{train}}$ refer to the minimum and maximum values in the training dataset, respectively. It is important to use the min/max values of the training set to prevent leaking information about the testing dataset to the model during training. Common choices for a and b are $a = 0$, $b = 1$ or $a = -1$ and $b = 1$. In this case, the latter was chosen.

3.2. Data-Driven Remaining Useful Life Prediction

In data-driven RUL prediction, the aim is, as the name suggests, use data to develop models for predicting the RUL of a system. Specifically, in this work, the data-driven models are based on machine learning. When using machine learning for RUL prediction, the problem is formulated as a supervised machine learning problem. In this case the goal is to learn a mapping f from measurements of the system over time, $\mathbf{x}_t^{(i)}$ to the corresponding RUL $y_t^{(i)}$:

$$f \colon \mathbf{x}_t^{(i)} \mapsto y_t^{(i)} \tag{2}$$

for $t = 0, \ldots, T^{(i)}$, and $i = 1, \ldots, N$ where $T^{(i)}$ is the life length of unit (also referred to as the end-of-life (EoL) of device/asset i) i and N is the number of devices in the full dataset. When f is parameterized by some parameters θ and a loss function can be defined $\mathcal{L}\left(f\left(X;\theta\right), y\right)$ the supervised learning problem can be stated an optimization problem:

$$\min_{\theta} \mathcal{L}\left(f\left(X;\theta\right), y\right). \tag{3}$$

In the context of deep artificial neural network (NN), the training is performed using gradient descent, through the backpropagation algorithm. As RUL prediction, in the context of ML, is a regression problem, a common choice for loss function is the mean squared error (MSE) loss or the square root of MSE, root mean squared error (RMSE) loss, defined as

$$\mathcal{L}_{\text{MSE}}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \hat{y}^{(i)} \right)^2 \tag{4}$$

and

$$\mathcal{L}_{\text{RMSE}}(y, \hat{y}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \hat{y}^{(i)} \right)^2}, \tag{5}$$

respectively. To penalize models for predicting poorly on outliers and minimize the error of the worst predictions, the so called $\mathcal{L}^\infty$ loss can be used, which is defined in the following manner:

$$\mathcal{L}^\infty(y, \hat{y}) = \max |y - \hat{y}|. \tag{6}$$

The loss function used when training the models is a combination of MSE and $\mathcal{L}^\infty$, namely

$$\mathcal{L}(y, \hat{y}) = L_{\text{MSE}}(y, \hat{y}) + \mathcal{L}^\infty(y, \hat{y}). \tag{7}$$

MSE and $\mathcal{L}^\infty$ is combined, because they serve two different purposes. MSE is a common regression loss function and is often used in RUL prediction. MSE loss is sensitive to outliers and will penalize predictions that are far off the correct RUL, thus leading to predictions that are more conservative, a property that is often desired in RUL prediction models. The $\mathcal{L}^\infty$-loss is used to further penalize the worst prediction of the mini-batch. The worst prediction is added to the final loss, further driving the model from large incorrect predictions, resulting in a conservative model.

3.3. TO Approach

The TO approach is based on training machine learning models on the entire time series data, such that the models can learn to capture the dynamics of the degradation process. Formally, given the original dataset of the form

$$\mathcal{D} = \left\{ \left(\mathbf{x}_t^{(i)}, y_t^{(i)} \right)_{t=0}^{T^{(i)}} \right\}_{i=1}^{N} \tag{8}$$

where $\mathbf{x}_t^{(i)} \in \mathbb{R}^d$ is the d number of sensor readings from the system at time t for asset $i = 1, \ldots, N$, and $y_t^{(i)}$ is the corresponding RUL at time $t = 0, \ldots, T^{(i)}$. Here N is the number of assets in the dataset, in this case, the number of TIDs in the dataset and $T^{(i)}$ is the EoL for asset i. In the TO approach, the machine learning models are trained to predict RUL at time t, by feeding all time series data up until time t to the model, i.e., a new dataset is constructed with tuples of the form:

$$\mathcal{T} = \left\{ \left(\left\{ \mathbf{x}_\tau^{(i)} \right\}_{\tau=0}^{t}, y_t^{(i)} \right)_{t=0}^{T^{(i)}} \right\}_{i=1}^{N}. \tag{9}$$

The dataset in Equation (9), is referred to the "trajectory dataset" because $\left\{ \mathbf{x}_\tau^{(i)} \right\}_{\tau=0}^{t}$ is the degradation trajectory of asset i up until time t measured by the sensor signals $\mathbf{x}_\tau^{(i)}$ for each time step $\tau = 0, \ldots, t$. These trajectories constitute the training data for the TO machine learning models.

3.3.1. Deep Neural Network Models

The models chosen for RUL prediction are variants of deep neural networks (LeCun, Bengio, & Hinton, 2015; Goodfellow, Bengio, & Courville, 2016). In particular, they are based on two main components, multilayer perceptron (MLP) (Haykin, 1994; Cybenko, 1989; Hornik, Stinchcombe, & White, 1989), and a type of recurrent neural network (RNN) called long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997). These models are suitable for sequential data such as time series, and in particular datasets of the form given by Equation (9).

MLP is the most fundamental type of artificial neural network. It consists of an input layer, hidden layers, and an output layer. As data are passed through each of the layers in the network, they are transformed non-linearly using so-called *activation functions*. The forward pass of the MLP is described by:

$$\begin{aligned}
h_i^{(1)} &= g^{(1)} \left(\sum_{j=1}^{n} w_{ij}^{(1)} x_j + b_i^{(1)} \right), \\
h_i^{(\ell)} &= g^{(\ell)} \left(\sum_{j=1}^{n} w_{ij}^{(\ell)} h_j^{(\ell-1)} + b_i^{(\ell)} \right), \ \ell = 2, \ldots, L-1, \\
y_i &= g^{(L)} \left(\sum_{j=1}^{n} w_{ij}^{(L)} h_j^{(L-1)} + b_i^{(L)} \right),
\end{aligned}$$

where $h_i^{(\ell)}$ is the i:th neuron in the ℓ:th layer, $g^{(\ell)}(\cdot)$ is the activation function between after layer ℓ. Furthermore, $w_{ij}^{(\ell)}$ is the so-called *weight matrix* containing the learnable weights connecting layer ℓ with layer $\ell - 1$.

LSTM networks are a type of RNN that are capable of learning long-term dependencies in sequence prediction problems. The key to an LSTM's ability to retain information over long periods is its unique architecture, which includes gates that regulate the flow of information.

The LSTM unit consists of several components, which are

governed by a set of equations. These components include the cell state (c_t), the hidden state (h_t), the input gate (i_t), the forget gate (f_t), the output gate (o_t), and the candidate cell state ($\tilde{c}_t$). The equations that govern these components are as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{10}$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{11}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{12}$$
$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{13}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{14}$$
$$h_t = o_t \odot \tanh(c_t), \tag{15}$$

where, denoting d and h as the dimension of the input and the number of hidden units, respectively:

- $x_t \in \mathbb{R}^d$ is the input at the current time step,
- $h_{t-1} \in (-1, 1)^h$ is the hidden state from the previous time step,
- $W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$, and $b \in \mathbb{R}^h$ are the weight matrices and bias vectors for the respective gates and candidate cell state,
- σ represents the sigmoid activation function, $\sigma(x) = (1 + \exp\{-x\})^{-1}$,
- $\tanh$ represents the hyperbolic tangent activation function,
- $\odot$ denotes element-wise multiplication (the Hadamard product).

Equations (10) to (15), have the following interpretations:

- Forget Gate $f_t \in (0, 1)^h$, Equation (10): Decides what information from the previous cell state should be forgotten.
- Input Gate $i_t \in (0, 1)^h$, Equation (11): Decides what new information will be stored in the cell state.
- Output Gate $o_t \in (0, 1)^h$, Equation (12): Decides what part of the cell state should output as the hidden state.
- Candidate Cell State $\tilde{c}_t \in (-1, 1)^h$, Equation (13): Creates a vector of new candidate values that could be added to the cell state.
- Cell State $c_t \in \mathbb{R}^h$, Equation (14): The actual cell state that keeps the long-term memory at the current time step t.
- Hidden State h_t, Equation (15): The final hidden state of the LSTM, the output of the LSTM unit.

These interdependent equations help the LSTM manage the flow of information and maintain relevant information over time, addressing the vanishing gradient problem that can occur in ordinary RNNs.

3.3.1.1. MLP-LSTM-MLP Model The multilayer perceptron long short-term memory multilayer perceptron (MLP-LSTM-MLP) architecture has, as the name suggests, three main components: an initial MLP that transforms time series through a non-linear mapping, followed by an LSTM which can learn long-term the relationship between time steps, and a final MLP that transforms the output of the LSTM. The MLP-LSTM-MLP model architecture was introduced by (Chaoub, Voisin, Cerisara, & Iung, 2021). In this paper, the model was used for RUL prediction of simulated turbofan jet engines. In Figure 4, a schematic of the architecture is shown. The first part of the model is an MLP which takes as input, the sensor signals for asset i at each time step $\mathbf{x}^{(i)}(t)$ where $i = 1, \ldots, N$ and $t = 0, \ldots, T^{(i)}$. The sensor signal is transformed by the MLP initial and fed into the LSTM. The output of the LSTM (the hidden state of the h_t) is fed into the final MLP which finally outputs the predicted RUL $y^{(i)}(t)$ for each time step. The hidden state of the LSTM is carried over to the next prediction of the LSTM, which enables the model to remember long-term relations in the time series data, a feature that is essential for prognostics.

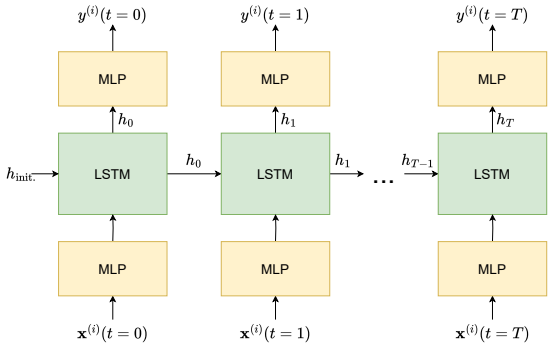


Figure 4. The MLP-LSTM-MLP architecture. An initial MLP takes the sensor signals at each time step as input. This is then fed into the LSTM model which outputs its hidden state. The hidden state of the LSTM is fed into the final MLP which outputs the predicted RUL.

3.3.1.2. Residual MLP-LSTM-MLP Model The residual MLP-LSTM-MLP (ResMLP-LSTM-MLP) is a slight modification to the MLP-LSTM-MLP architecture, where the initial MLP is substituted with a residual MLP. The residual MLP consists of several residual MLP blocks which is an MLP with a residual skip connection. If $\mathbf{x}$ denotes the input data and $\mathcal{F}_{\text{MLP}}(\cdot)$ denotes the transformation of the MLP, a residual skip connection is simply

$$\mathcal{F}_{\text{ResMLPBlock}}(\mathbf{x}) = \mathcal{F}_{\text{MLP}}(\mathbf{x}) + \mathbf{x}. \tag{16}$$

Residual neural network (ResNet) was first described in (K. He, Zhang, Ren, & Sun, 2016), and was invented to allow information to propagate more easily in deep neural net-

works. ResNets allows for creating deeper neural networks while mitigating some issues with very deep neural networks such as vanishing/exploding gradients. Deeper neural networks can potentially allow the model to learn better representations of the data which leads to better performance. In Figure 5 the ResMLP-LSTM-MLP architecture is shown in a schematic. The initial step of the model uses a series of MLPs with residual skip connections, forming residual blocks. There are n blocks in the model. Similarly to the MLP-LSTM-MLP model, this part of the model takes sensor measurements at each time step as input. Then, the model is identical to the MLP-LSTM-MLP model, with an LSTM for modeling sequential data and a final MLP for outputting RUL. See paragraph 3.3.1.1 for details.

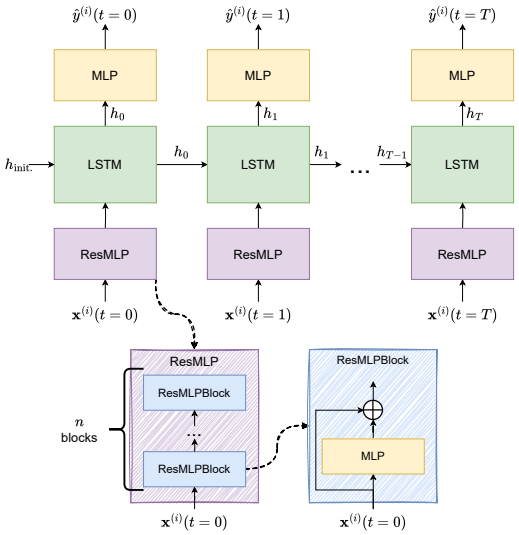


Figure 5. The ResMLP-LSTM-MLP architecture.

3.3.2. Model Training Procedure

The models are trained using gradient descent on the loss function, cf. Equation (7), and specifically, for deep neural networks, the algorithm is called backpropagation (Rumelhart, Hinton, & Williams, 1986). The models are implemented using the Python programming language (Van Rossum & Drake, 2009) and the deep learning framework PyTorch (Paszke et al., 2019). When training the models, a number of so-called *hyperparameters* need to be specified. A hyperparameter is not a parameter of the model that is changed during training but still has an effect on the training of the model. As such, it is important to find good hyperparameters for the model so that it converges. To achieve this, one often employs *hyperparameter search*, a method for finding suitable hyperparameters. The hyperparameters search is implemented using the "ray" Python library (Liaw et al., 2018). In particular, the Asynchronous Hyperband Scheduler is used, a method for massively parallel hyperparameter search, described in (Li et al., 2020). In Table 2 the hyperparameter settings for each model is presented.

3.3.3. Uncertainty Estimation

Uncertainty estimation in deep neural network models can be achieved through various means and techniques. One such technique is called "Monte Carlo dropout" and was introduced in (Gal & Ghahramani, 2016). Monte Carlo dropout relies on a deep learning regularization technique called "dropout", first described in (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). Dropout is applied to layers in a deep neural network, and means that some neurons in the layer are randomly dropped out with some probability $p \in (0, 1)$. This means that these neurons are zeroed-out in the forward pass and therefore do not contribute. As a consequence, the network cannot rely on particular neurons to be activated during the forward pass and needs to learn redundant representations of the data This has several potential benefits:

- Reducing overfitting,
- Increasing robustness,
- More efficient representations.

Usually, when the training is completed, the dropout probability is set to $p = 0$, such that all neurons are activated at inference time. However, one can leverage the natural variance induced by keeping $p > 0$ at inference time and interpret this variance as an estimation of the uncertainty of the model. This is the core idea of Monte Carlo dropout uncertainty estimation. At inference time, the dropout probability is kept at the same level as during training, and k forward passes are made to achieve a spread in predictions. By calculating the mean of the predictions, the "mean prediction" is constructed as a point estimate of the model. In addition, assuming the predictions of the model are normally distributed, the 95% confidence interval is calculated in the following manner:

$$I_{95\%} = (\mu - 1.96\,\sigma, \mu + 1.96\,\sigma) \tag{17}$$

where μ and σ are the mean and the standard deviation of the predictions of the model, respectively. 1.96 is the 97.5th percentile of the standard normal distribution, meaning 95% of the area under the standard normal distribution curve lies in the interval $[-1.96, 1.96]$. Dropout is applied to the final layers of the model, specifically in the final MLP of each model.

3.4. FO Approach

The FO approach focuses on transforming the data into an aligned two-dimensional (tabular) problem so that compatible ML-methods can be implemented.

The work of (Wold, Kettaneh-Wold, MacGregor, & Dunn, 2009) is seminal in advancing feature-based ML for industrial

Hyperparameter	MLP-LSTM-MLP	ResMLP-LSTM-MLP
Number of epochs	500	500
Batch size	2	2
Effective batch size	28	28
Initial MLP hidden sizes	[128, 64, 64]	[128, 64, 64]
LSTM hidden size	32	32
Output MLP hidden sizes	[64, 32]	[64, 32]
Learning rate	0.001	0.001
Optimizer weight decay	0.001	0.001
Dropout rate	0.5	0.5
Number of residual blocks	N/A	4

Table 2. The hyperparameters for the two models are presented above.

processes, laying the foundation for what is now referred to as batch data analytics (BDA), and serving as a cornerstone in the evolution of data-driven industrial analytics.

Furthermore, (Rendall, Chiang, & Reis, 2019) provides an insightful and comprehensive review of the modeling complexities and implementation challenges associated with these methods, offering a nuanced understanding of their intricacies and practical applicability.

The works of (Mählkvist, Ejenstam, & Kyprianidis, 2022, 2023) have demonstrated the effective implementation of a feature-based approach within industrial systems, highlighting the applicability and efficiency of data-driven methodologies in optimizing process performance.

3.4.1. Data Pre-processing

There are two steps to the pre-processing of the FO approach. These are, in order of implementation, feature generation and missing value removal.

Feature generation, is step based on statistical pattern analysis (SPA), relies on the works of (Wang & He, 2010; Q. P. He & Wang, 2011) and extracts the mean, variance, skewness, and kurtosis. Mean, variance, skewness, and kurtosis collectively provide a comprehensive characterization of a dataset's distribution. The mean indicates the central tendency, offering a measure of the dataset's average value. Variance measures the dispersion, showing how much the data points spread around the mean, which helps in understanding the variability within the dataset. Skewness assesses the asymmetry of the distribution, revealing whether data points are more concentrated on one side of the mean. Kurtosis evaluates the 'tailedness' of the distribution, indicating the presence of outliers and the sharpness of the data peaks. Together, these metrics offer a detailed summary of the central value, spread, symmetry, and extremity, thereby providing a robust indication of the dataset's characteristics.

In data preprocessing, missing values are handled by setting a threshold to determine which features to retain. In this work, features with less than 75% of their values present are removed. After excluding these features, all rows containing any remaining missing values are also removed. This process ensures that the dataset used for training the models in the FO approach is fully populated and complete, which is essential, as these models cannot be trained on datasets with missing values.

3.4.2. Model Selection

For the FO approach, two models are used. The two models possess two defining characteristics that are relevant when determining which model works best for any dataset. First, the models can be either parametric or non-parametric. Second, can the models accommodate non-linearity. For the FO approach: RFR and SVR are used, with the former being non-parametric and the latter being parametric and both being able to accommodate non-linear data.

3.4.2.1. RFR RFR, also known as random forest regression, is an ensemble learning method used for regression tasks. This technique functions by constructing a multitude of decision trees during the training phase. Each tree in the forest relies on the values of a random vector, which is sampled independently and follows the same distribution across all trees (Breiman, 2001). The parameters for RFR include the maximum number of features and the number of estimators, which represent the number of trees in the forest. The selection of the number of trees is crucial as it affects the model's performance, with a larger number of trees often providing better generalization at the cost of increased computational time

3.4.2.2. SVR SVR enhances the traditional support vector machine regressor by employing kernels to expand the feature space, thereby accommodating non-linear characteristics (Boser, Guyon, & Vapnik, 1992; James, Witten, Hastie, & Tibshirani, 2013). radial basis function (RBF) kernel, also known as the Gaussian kernel, maps the input features into an infinite-dimensional space. It measures the similarity between data points based on their distance, allowing the capture of complex, non-linear relationships. The RBFkernel

is particularly powerful for handling data that is not linearly separable (James et al., 2013).

3.4.3. Hyperparameter estimation

A train-test split is employed to ensure training is conducted without data leakage. In this process, the dataset is divided into two distinct subsets: the training set and the test set. The training set is used exclusively to train the model, allowing the algorithm to learn patterns and features without any influence from the test data. The test set, kept entirely separate, is utilized to evaluate the model's performance, providing an unbiased measure of its ability to generalize to unseen data. This separation is crucial for preventing data leakage, which can lead to overly optimistic performance estimates and poor generalization. By employing a train-test split, the integrity of the model evaluation process is maintained, ensuring accurate and reliable assessment of the model's true predictive capability.

Hyperparameter estimation is performed using random grid search, where parameters are selected randomly from a list or defined distribution over a set number of iterations. For each model, 100 search iterations are conducted. Furthermore, (Bergstra & Bengio, 2012) demonstrates the benefits of the random approach, highlighting its ability to reduce computational resources while achieving results that are equal or superior to those obtained through the conventional exhaustive grid search.

The log-uniform distribution is used to define the hyperparameter range for many of the parameter is particularly useful for parameters that cover several orders of magnitude, as it helps in exploring a wide range of scales effectively. The log-uniform distribution is defined as

$$\mathcal{U}(x; a, b) = \begin{cases} \dfrac{1}{x\left[\ln b - \ln a\right]} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

where $\mathcal{U}$ is the log-normal distribution and a, b are the lower and upper bound, respectively.

Similarly, when dealing with hyperparameters that require discrete integer values within a specific range, the random integer distribution is employed. This distribution is particularly beneficial in exploring discrete parameter spaces effectively. The random integer distribution is defined as

$$\mathcal{R}(k; m, n) = \frac{1}{n - m + 1} \tag{19}$$

for any $k = m, m + 1, \ldots, n$. $\mathcal{R}$ is the uniform distribution and m, n are the lower and upper bound, respectively, representing the inclusive range of possible integer values.

The hyperparameter distribution for the FO models is displayed in Table 3.

Table 3. Hyperparameter distributions

Model	Parameter	Scope
RFR	Number of Estimators	$\mathcal{R}(10, 1000)$
RFR	Maximum Depth	$\mathcal{R}(1, 100)$
RFR	Maximum Features	$\mathcal{R}(1, 100)$
SVM	C	$\mathcal{U}(0.1, 1000)$
SVM	γ	$\mathcal{U}(0.001, 1)$
SVM	ϵ	$\mathcal{U}(0.01, 0.1)$

3.4.4. Resulting hyperparameters

The hyperparameters of the best models are shown in Table 4.

3.5. Evaluation Metrics

The main evaluation metrics are the RMSE (defined in Equation (5)), mean absolute error (MAE) defined as

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|, \tag{20}$$

and coefficient of determination (R^2) defined as

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}, \tag{21}$$

where

$$SS_{\text{res}} = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2, \tag{22}$$

$$SS_{\text{tot}} = \sum_{i=1}^{n} \left(y_i - \bar{y}\right)^2. \tag{23}$$

Here, $\bar{y}$ is the mean of the model outputs. The RMSE and MAE are non-negative real numbers where an RMSE/MAE of zero means a model that perfectly agrees with the ground truth. RMSE is penalizing outliers more heavily whereas the MAE is a linear measure is more lenient on large deviations. The R^2 measures the proportion of variation in the model vs. the variation in the data. A model that outputs the mean of the true target will get an R^2-score of zero, whereas a perfect

Table 4. Hyperparameter results

Model	Parameter	Best Model
RFR	Number of Estimators	155
RFR	Maximum Depth	16
RFR	Maximum Features	46
SVM	C	0.24
SVM	γ	0.1596
SVM	ϵ	0.089

Table 5. Model evaluation using RMSE, R^2, and MAE metrics. NN and ResNN are placeholder names for the lengthy MLP-LSTM-MLP and ResMLP-LSTM-MLP models.

Model	RMSE	R^2	MAE
NN	0.131781	0.592906	0.080010
ResNN	0.130005	0.603803	0.078494
RFR	**0.105280**	**0.740174**	0.078300
SVM	0.105383	0.739666	**0.074822**

model will get a score of one. The R^2 can be negative since the model can be arbitrarily bad.

4. RESULTS & DISCUSSION

4.1. Comparison of Machine Learning Model Performance

This section contains a detailed comparison of the MLP-LSTM-MLP, ResMLP-LSTM-MLP, RFR, and SVR models in terms of accuracy and reliability in predicting RUL. Comparative analysis showed that each model, influenced by the nature of the pre-processed data, offered varying degrees of accuracy in RUL predictions. The MLP-LSTM-MLP and ResMLP-LSTM-MLP are designed for sequence data, and RFR and SVR, tailored for statistical features, demonstrated distinct performance profiles.

The RMSE score of each model is presented in Table 5. It shows that the FO-models outperform the TO-models, with RFR being the best performing model. The assumption in the FO-approach seems to have a beneficial impact on the models' interpretation of the data.

The relationship between actual and predicted RUL can be observed in Figure 6, which contains four subplots, one subplot per model, and shows how the predicted RUL for the test batches differ from the actual RUL. On the horizontal axis, the TIDs are shown sorted in descending order according to true RUL. The vertical axis, the RUL is shown, where the blue solid markers represent the actual RUL and the x-shaped orange markers represent the predictions made by the various models. One can observe that the FO based models are closer to the true predictions and slightly more likely to underestimate RUL as compared to the TO based models. The TO based models also seem to be more sensitive to outliers, with more predictions far from the true RUL.

In order to make comparison more tangible, Figure 7 was designed to show the sample volume and distribution over bins of actual RUL for all models. The figure consists of two subplots, the upper figure Figure 7a, showing the sample volume of the bin, and the lower figure Figure 7b, depicting a boxplot for each model. Each bin is of size 0.1, starting from 0 and incrementally increasing up to 0.7, resulting in 7 intervals. Each interval contains the samples that have their actual RUL with that interval. Each model has a box in each interval describing the distribution of that model in that interval. The top-subplot show an overview with a wider y-axis, while the bottom-subplot focuses in on a narrow, more relevant, part of the y-axis. The y-axis show the distribution of the RUL residual of the model in that interval.

Initially, from high actual RUL, the TO models have a residual closer to 0, compared to the FO models (looking at intervals from 0.4 to 0.7). At this stage, the FO models catch up, and have comparable performance from 0.2 to 0.4. In the interval from 0.1 to 0.2, the FO models have superior performance compared to the TO models. In the final interval, from 0 to 0.1, the FO models are generally overestimating RUL with relatively low spread. On the other hand, the TO models are also overestimating, although to a lesser extent, and has a larger spread.

Looking at Figure 7, it is challenging to discern that the FO outperform the TO, without referencing Table 5. Looking at the number of samples in the intervals, the FO models perform better in the high population intervals (such as interval 0.1 to 0.2), while the TO models perform better in low population intervals (such as interval 0.5 to 0.6).

4.1.1. Outlier Sensitivity

As can be observed in Figure 6, the models exhibit different behavior regarding outliers. E.g., the most deviating prediction for MLP-LSTM-MLP and ResMLP-LSTM-MLP is TID 638 which can be observed in the top right corner in Figure 6. The impact of TID 638 on RFR and SVR predictions is not nearly as severe. To assess the sensitivity to outliers, one can observe in Table 6that if TID is removed from the dataset, the evaluation results change drastically.

Model	RMSE	R^2	MAE
NN	0.106114	0.733590	0.073870
ResNN	0.104815	0.740072	**0.072448**
RFR	0.105484	0.736744	0.078348
SVM	**0.101302**	**0.757203**	0.072777

Table 6. Evaluation of the models without TID 638. The evaluation metrics are the RMSE, coefficient of determination R^2, and MAE. Here, NN and ResNet are shorthand notations for the rather lengthy MLP-LSTM-MLP and ResMLP-LSTM-MLP model names, respectively.

4.1.2. Uncertainty Estimation Results

As mentioned in Section 3.3.3, the TO-models are endowed with uncertainty estimation capabilities, through Monte Carlo Dropout. In the final evaluation, the uncertainty estimation is transformed into a point-estimate for predicted RUL by taking the mean of all predictions. For each sample, ten forward passes are used to create a spread of predictions. The model uses a dropout rate of 20 $\%$ in the last MLP-layer. In Fig-

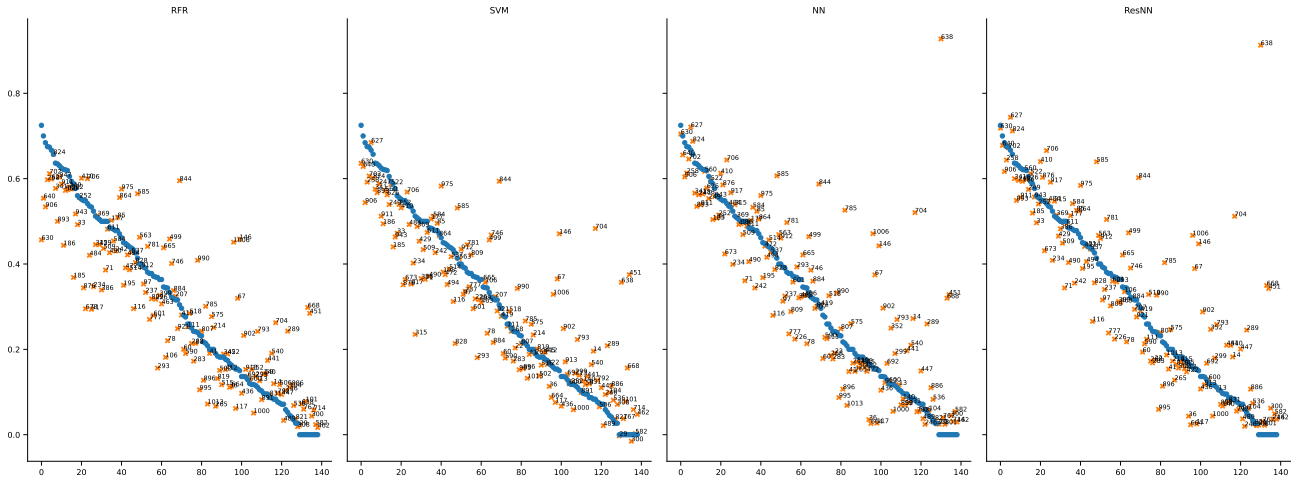


Figure 6. RUL predictions for the various models. On the horizontal axis, the TIDs sorted by RUL in descending order are shown. On the vertical axis RUL is shown, the true RUL is shown as a solid line and the predictions are shown as scatter points.

ure 8 the RUL predictions for TID 13 (in the test set), with uncertainty estimation is shown, using the MLP-LSTM-MLP model.

5. CONCLUSIONS

In this paper, methods for RUL prediction of resistance heating wires undergoing cyclical testing is investigated. The paper explores two contrasting approaches; "feature-oriented (FO)" and "trajectory-oriented (TO)". In the FO approach, statistical measures are derived from the time series data in order to convert the data into a tabular form. This tabular dataset can then be used for training machine learning models suitable for tabular data. On the other hand, the TO approach trains the machine learning models on the full time series data, and thus needs models capable of handling such data. A natural choice is various deep learning techniques, in particular, variants of RNNs.

The results show that the FO based methods are performant and robust to outliers. In particular, the RFR model has the best test dataset performance according to the chosen evaluation metrics. The TO based methods are less biased across various levels of degradation, which is a desirable trait in PHM systems. The FO-based methods are, generally, systematically underestimating RUL at high true RUL and overestimating RUL at low actual RUL. This is still somewhat present in the TO based models but to a lesser degree.

Moreover, when removing one particularly strong outlier, one can observe that the RFR model is no longer outperforming the other models and the scores are much more closely aligned between the approaches.

These insights hold significant promise for enhancing predictive maintenance strategies and operational efficiency in industries relying on components like resistance heating wires.

ACKNOWLEDGMENT

This research work has been funded by the Knowledge Foundation within the framework of the ARRAY (Grant Number 20170214) and INDTECH (Grant Number 20200132) Research School projects, participating companies and Mälardalen University. The authors also give their thanks to the RISE internal project DIGIPROD for supporting this work.

REFERENCES

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, *13*(2).

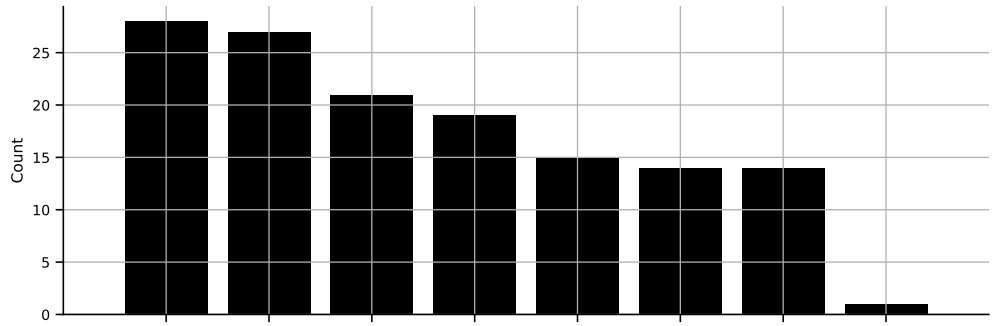
Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory* (pp. 144–152).

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32. (Publisher: Springer)

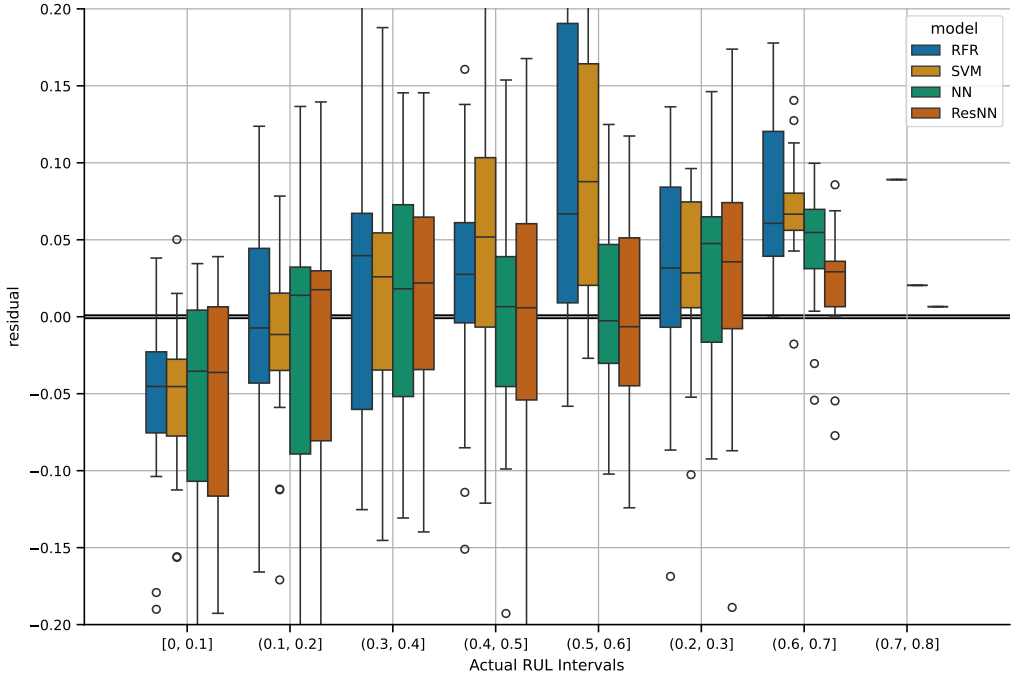
Chaoub, A., Voisin, A., Cerisara, C., & Iung, B. (2021). Learning representations with end-to-end models for improved remaining useful life prognostics. *CoRR*, *abs/2104.05049*. Retrieved from https://arxiv.org/abs/2104.05049

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*(4), 303–314. Retrieved from https://doi.org/10.1007/BF02551274 doi: 10.1007/BF02551274

Gal, Y., & Ghahramani, Z. (2016, 20–22 Jun). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd in-*



(a) The number of observations/samples in each bin in the below plot. Most test samples have 50 % RUL or less.



(b) The four models' predictions in eight intervals of width 0.1 (10 % RUL) on the x-axis. The y-axis represents the error (or residual) i.e., the difference between the true and predicted RUL, $e = y - \hat{y}$.

Figure 7. Models' RUL predictions within intervals of 0.1 width. The top plot shows the number of samples within each interval, and the below shows the prediction residual vs. actual RUL intervals.

ternational conference on machine learning (Vol. 48, pp. 1050–1059). New York, New York, USA: PMLR. Retrieved from https://proceedings.mlr.press/v48/gal16.html

Galar, D., Goebel, K., Sandborn, P., & Kumar, U. (2021). *Prognostics and remaining useful life (rul) estimation: Predicting with confidence* (1st ed.). CRC Press. Retrieved from https://doi.org/10.1201/9781003097242 doi: 10.1201/9781003097242

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (http://www.deeplearningbook.org)

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 ieee conference on computer vision and pattern recognition (cvpr)* (p. 770-778). doi: 10.1109/CVPR.2016.90

He, Q. P., & Wang, J. (2011). Statistics pattern analysis: A new process monitoring framework and its application to semiconductor batch processes. *AIChE Journal*, *57*(1), 107–121. Retrieved 2022-09-07, from http://onlinelibrary.wiley.com/doi/abs/10.1002/aic.12247 doi: 10.1002/aic.12247

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term

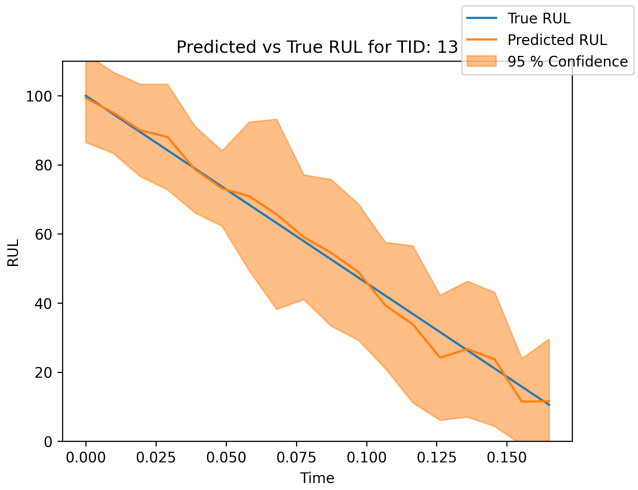


Figure 8. RUL predictions for TID 13 with uncertainty estimation using the MLP-LSTM-MLP model.

memory. *Neural computation*, *9*(8), 1735–1780.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359-366. Retrieved from https://www.sciencedirect.com/science/article/pii/0893608089900208 doi: https://doi.org/10.1016/0893-6080(89)90020-8

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112). Springer.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. Retrieved from https://doi.org/10.1038/nature14539 doi: 10.1038/nature14539

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B., & Talwalkar, A. (2020). *A system for massively parallel hyperparameter tuning.*

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., & Stoica, I. (2018). Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*.

Mählkvist, S., Ejenstam, J., & Kyprianidis, K. (2022, March). Consolidating Industrial Batch Process Data for Machine Learning. In (pp. 76–83). Retrieved 2022-12-10, from https://ecp.ep.liu.se/index.php/sims/article/view/330 doi: 10.3384/ecp2118576

Mählkvist, S., Ejenstam, J., & Kyprianidis, K. (2023). Cost-sensitive decision support for industrial batch processes. *Sensors*, *23*(23), 9464. (Publisher: MDPI)

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pecht, M., & Kang, M. (2018). *Prognostics and health management of electronics : fundamentals, machine learning, and internet of things* (Second edition. ed.). Hoboken, New Jersey: John Wiley & Sons.

Rendall, R., Chiang, L. H., & Reis, M. S. (2019, May). Data-driven methods for batch data analysis – A critical overview and mapping on the complexity scale. *Computers & Chemical Engineering*, *124*, 1–13. Retrieved 2022-11-11, from https://linkinghub.elsevier.com/retrieve/pii/S0098135418311104 doi: 10.1016/j.compchemeng.2019.01.014

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. Retrieved from https://doi.org/10.1038/323533a0 doi: 10.1038/323533a0

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958. Retrieved from http://jmlr.org/papers/v15/srivastava14a.html

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. Scotts Valley, CA: CreateSpace.

Wang, J., & He, Q. P. (2010, September). Multivariate Statistical Process Monitoring Based on Statistics Pattern Analysis. *Industrial & Engineering Chemistry Research*, *49*(17), 7858–7869. Retrieved 2022-09-16, from https://pubs.acs.org/doi/10.1021/ie901911p doi: 10.1021/ie901911p

Wold, S., Kettaneh-Wold, N., MacGregor, J., & Dunn, K. (2009). Batch Process Modeling and MSPC. In *Comprehensive Chemometrics* (pp. 163–197). Elsevier. Retrieved 2022-09-07, from https://linkinghub.elsevier.com/retrieve/pii/B9780444527011001083 doi: 10.1016/B978-044452701-1.00108-3

Yoro, K. O., & Daramola, M. O. (2020). Chapter 1 - CO2 emission sources, greenhouse gases, and the global warming effect. In M. R. Rahimpour, M. Farsi, & M. A. Makarem (Eds.), *Advances in carbon capture* (pp. 3–28). Woodhead Publishing. Retrieved from https://www.sciencedirect.com/science/article/pii/B9780128196571000013 doi: https://doi.org/10.1016/B978-0-12-819657-1.00001-3