# Data-Driven RUL Prediction Using Performance Metrics

**Abel Diaz-Gonzalez**
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

**Austin Coursey**
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

**Marcos Quinones-Grueiro**
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

**Chetan S. Kulkarni**
NASA Ames Research Center (KBR, Inc), Moffett Field, CA, USA

**Gautam Biswas**
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA

## Abstract

Prognostics is the scientific study of component and system degradation with use, and the prediction of when failure may occur. In this work, we propose a new data-driven method for predicting a system's remaining useful life (RUL) without needing an accurate system model or expert knowledge. Instead, we use system operational data to estimate how the system's performance metrics change with time. Although this is a purely data-driven approach, the method's design is inspired by model-based techniques. First, we frame a novel Multitask Machine Learning architecture to simultaneously learn the general pattern of performance degradation and the individual trajectories from run-to-failure performance trajectory data. We apply this method to the set of performance metrics that determine the system's end-of-life (EOL), building a performance trajectory library of the system operation under different operational conditions. We leverage the performance metric library as prior belief and develop a Bayesian deep learning approach to update the performance measures over time and predict the system EOL. We evaluate our method on two datasets of the N-CMAPSS benchmark, achieving satisfactory results in terms of overall performance and uncertainty estimation accuracy. Overall, our approach illustrates a generalized deep learning architecture that can more effectively predict the system RUL for a collection of identical systems.

**2012 ACM Subject Classification** Applied computing → Aerospace; Computing methodologies → Artificial intelligence; Computing methodologies → Machine learning approaches

**Keywords and phrases** remaining useful life, data-driven methods, machine learning, performance metric, multitask machine learning, Monte Carlo

**Digital Object Identifier** 10.4230/OASIcs.DX.2024.21

**Category** Short Paper

## 1 Introduction

Industrial systems and their components experience degradation over time with a risk of experiencing failure. To perform condition-based maintenance (CBM), operators must use prognostics approaches to estimate the remaining useful life (RUL) of these systems and their components. Accurate estimation of the RUL allows operators to make informed decisions about system health and maintenance decisions thus increasing the resilience of real-world system operations [15]. Techniques for estimating RUL can be divided into three major categories: model-based, data-driven, and hybrid [8].

Model-based techniques leverage system physics to develop mathematical models for prognostics, achieving notable success. For instance, [5] presents a method estimating state and damage progression parameters, then predicting the end of life (EOL) by propagating these estimates. Applied to a centrifugal pump model, they found the unscented Kalman filter optimal for state-parameter estimation, though other filters might suit different applications [6]. Recent advancements include extending classical physics-based prognostics with mechanistic approaches for accurate RUL estimation in lithium-ion batteries, albeit with higher computational costs [7]. These techniques are powerful when data is limited but require expert knowledge, which may be impractical for complex systems [4].

To overcome modeling challenges, data-driven techniques have been developed, learning mappings from measured variables to RUL without detailed expert input. Deep learning methods, particularly recurrent neural networks (RNNs), are popular in aircraft engine prognostics [19]. For instance, [9] employs a bidirectional long short-term memory network on the C-MAPSS dataset. However, RNNs struggle with long sequences due to the vanishing gradient problem [14], and even advanced RNNs lack parallelization [18]. Transformers, like the temporal flow transformer for bearing RUL estimation [3] and methods predicting proton exchange membrane fuel cell degradation [11], have shown superior performance. Other popular approaches include convolutional neural networks fused with LSTMs [12], Gaussian processes [2], and multi-layer perceptrons [17].

Data-driven methods, despite their popularity, have limitations such as lack of explainability [8] and the need for large, representative datasets [4]. Addressing these, [21] and [13] incorporate uncertainty quantification in RUL estimation. Hybrid approaches combining data-driven and model-based methods have been proposed, such as fusing data-driven features with degradation models [20] or using physics-derived features in deep learning models [4].

This paper introduces a novel approach inspired by model-based prognostics [10] but adopts an entirely data-driven approach. We reformulate model-based components using data-driven models, offering explainability without requiring expert knowledge. Our method frames performance degradation modeling as a multitask machine-learning problem, creating a data-driven performance trajectory library. This library is used to predict EOL by solving the uncertainty propagation problem for each performance metric and updating prior beliefs with new data. Evaluated on two N-CMAPSS sub-datasets for aircraft engine prognostics [1], our contributions are:

- An end-to-end data-driven prognostics approach estimating RUL and associated uncertainty based on performance metrics, independent of system physics.
- Demonstration of the framework's potential on N-CMAPSS datasets, achieving satisfactory predictive results and outlining future research directions.

The rest of the paper is structured as follows. Section 2 provides an explanation and the theoretical foundations of the proposed method. Subsection 2.2 focuses on the training process, while the second step, responsible for predicting the system's RUL, is addressed in Subsection 2.3. Section 3 presents an analysis of the experimental results and Section 4 concludes with a summary of the main findings and outlines future work.

## 2 Proposed Approach

Our proposed approach consists of two main steps. As a first step, we learn a library of run-to-failure trajectory distributions for each performance metric using training data. We create a new *Multitask Machine Learning model* for each performance metric. Each training trajectory corresponding to a performance metric constitutes a learning task, and all trajectories share the general performance pattern as common information. Learning this library constitutes the offline training phase of our method as shown in Figure 1a.

Our second step uses the learned trajectory distribution libraries as prior beliefs for predicting the end of life (EOL) corresponding to each performance metric. This is illustrated in Figure 1b. The procedure to generate the predictions associated with each performance metric is shown in Figure 1c and can be summarized as follows:

1. **First Prediction:** At time step $n = 0$, before receiving observations, we make an initial prediction for each performance metric by sampling the performance trajectories in our library uniformly $n_{\text{paths}}$ times, where $n_{\text{paths}}$ is a hyperparameter of the method indicating the number of predictions we make at each time step. Then, we compute the EOL and uncertainty boundaries of the system RUL from these initial predictions.
2. For each time step $n \geq 1$ repeat:
   a. **Performance Prediction.**
      For each Performance metric $i = 1, 2, \ldots, \kappa$, repeat:
      i. **Correction Step (see Section 2.3.4):** Select the best predictions from the previous time step in the light of the new observation.
      ii. **Build Trajectory Belief (Section 2.3.4):** Select a set of beliefs, or a guess, around the best performance predictions obtained in the previous step.
      iii. **Uncertainty Propagation (Section 2.3.3):** For each belief, predict the performance by finding the best linear combination of the learned trajectory library that maximizes the posterior probability under the given prior belief. The set of optimal linear combinations for each belief gives the set of $n_{\text{paths}}$ predicted performance trajectory at time $n$.
   b. **Compute System RUL(Section 2.3.5):** After computing the $n_{\text{paths}}$ performance predictions for each performance metric, we compute the EOL and uncertainty bounds for the system at the current time.

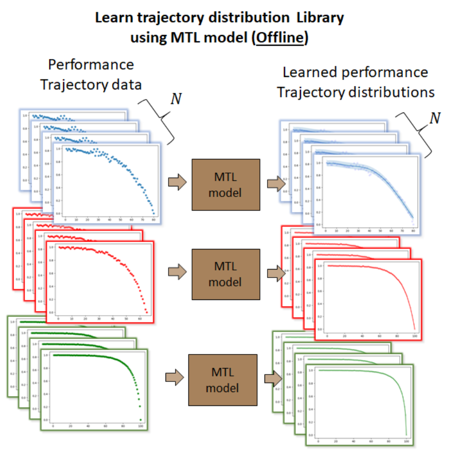
## 2.1 Training data

Denote by $\kappa$ the number of performance metrics that determine the EOL of the system. For simplicity, we assume that we record the value of every performance metric throughout its life. Then, if we have the run-to-failure trajectories of $N$ units, our training data consists of the set of $N$ (multivariate) time series of performance metric trajectories $\{u_i\}_{i=1}^{N}$, where $u_i = \{(t_j, p_j)\}_{j=1}^{L_i}$, $p_j = (p_j^{(1)}, p_j^{(2)}, \ldots, p_j^{(\kappa)})$ is the vector of performance metric values collected at time $t_j$, and $L_i$ denotes the number of time steps until the EOL of the $i^{th}$ training unit. Notice that we are omitting the unit indexation to simplify the notation. Our approach also needs a maximum life value $t_{\max}$ that determines the maximum life we are considering. Then $t_1 < t_2 < \cdots < t_{L_i} \in [0, t_{\max}]$ for every training unit and the EOL of the $i^{th}$ unit is $\text{EOL}_i = t_{L_i}$ for $i = 1, 2, \ldots, N$.

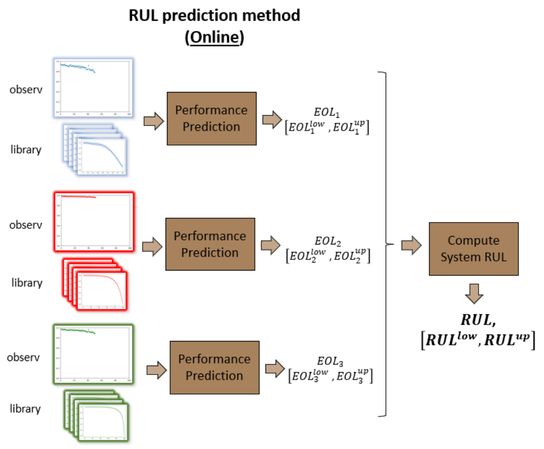
We also assume that the thresholds determining the EOL for each performance metric are constant values independent of time. Therefore, by scaling, we can assume without loss of generality that the performance metrics are decreasing functions from $[0, t_{L_i}] \subset [0, t_{\max}]$ to $[0, 1]$, where a value of 1 indicates a completely healthy performance indicator and a value of 0 means that the performance metric has reached its EOL.

## 2.2 Learning the Performance Trajectory Library

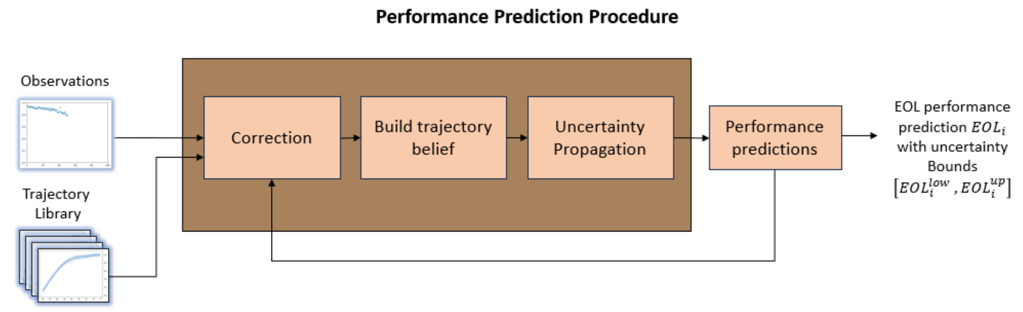
During the training process, we learn all the run-to-failure trajectories of the same performance metric in parallel (as individual Gaussian stochastic processes) by using a multitask machine learning architecture that learns the performance metric pattern in a block shared along the


Learn trajectory distribution Library using MTL model (Offline)
Performance Trajectory data
N
MTL model
MTL model
MTL model
Learned performance Trajectory distributions
N

**(a)** Training step.


RUL prediction method (Online)
observ
library
Performance Prediction
$EOL_1$ $[EOL_1^{low}, EOL_1^{up}]$
observ
library
Performance Prediction
$EOL_2$ $[EOL_2^{low}, EOL_2^{up}]$
observ
library
Performance Prediction
$EOL_3$ $[EOL_3^{low}, EOL_3^{up}]$
Compute System RUL
$RUL$, $[RUL^{low}, RUL^{up}]$

**(b)** RUL prediction step.


Performance Prediction Procedure
Observations
Trajectory Library
Correction
Build trajectory belief
Uncertainty Propagation
Performance predictions
EOL performance prediction $EOL_i$ with uncertainty Bounds $[EOL_i^{low}, EOL_i^{up}]$

**(c)** Performance prediction procedure.

**Figure 1** Overview of the proposed method for the case of three performance metrics $\kappa = 3$.

individual trajectory models. Specifically, for each performance metric with $N$ run-to-failure trajectories, we learn a collection of time series $h_i(t) = (\mu_i(t), \sigma_i(t))$ for $i = 1, 2, \ldots, N$, where $\mu_i(t)$ and $\sigma_i(t)$ represent the mean and standard deviation of the $i^{th}$ run-to-failure trajectory at time $t$. We call this collection of time series $h$ a trajectory library. We model the individual trajectories by using linear transformations of inputs $g^{\text{in}} = (g_1^{\text{in}}, g_2^{\text{in}}, \ldots, g_N^{\text{in}})$ and outputs $g^{\text{out}} = (g_1^{\text{out}}, g_2^{\text{out}}, \ldots, g_N^{\text{out}})$ of the common block, from now on referred to as latent function $f$ in charge of learning the performance pattern. By composing both, we build an individual model for the performance trajectory distribution of each training unit: $h_i(x) = (g_i^{\text{out}} \circ f \circ g_i^{\text{in}})(x), \quad i = 1, 2, \ldots, N$. The functions $g_i^{\text{out}}$, $g_i^{\text{in}}$, and $f$ are learnable. Note that the latent function $f$ and its learnable parameters are common to all the trajectory models $h_i$. Thus, by learning from all the trajectories, we ensure that the individual trajectory information passes through the latent function $f$. This enforces $f$ to learn only the common information across trajectories while simultaneously transferring information from long to short trajectories.

We modeled our latent function $f$ as a multilayer perceptron (MLP) neural network with residual connections. The functions $g_i^{\text{in}} : \mathbb{R} \to \mathbb{R}$ and $g_i^{\text{out}} : \mathbb{R}^2 \to \mathbb{R} \times \mathbb{R}_+$ are linear layers $g_i^{\text{in}}(x) = a_i x + b_i$ and $g_i^{\text{out}}(x_1, x_2) = (\mu(x_i), \sigma(x_2)) = (c_i^{\text{mean}} x_1 + d_i^{\text{mean}}, c_i^{\text{std}} x_2 + d_i^{\text{std}})$, with learnable parameters $a_i, b_i, c_i^{\text{mean}}, c_i^{\text{std}}, d_i^{\text{mean}}, d_i^{\text{std}}$ for $i = 1, 2, \ldots, N$. Since the standard deviation is a positive value, we also need to apply a positive function to the second output of $g_i^{\text{out}}$. In our implementation, we chose the Softplus function ($\text{Softplus}(x) = \log(1 + e^x)$). To introduce monotonicity in the performance metric, we restrict the sign of the slope coefficients of the linear transformation layers. All slopes are positive except for the slope of the output layer associated with the mean, whose sign depends on the monotonicity of our performance metric. Assuming our performance is monotonically decreasing, we restrict the slope coefficients as $a_i, c_i^{\text{std}} > 0$ and $c_i^{\text{mean}} < 0$ for $i = 1, 2, \ldots, N$. We also introduce monotonicity in our latent function $f$ by adding a term in the loss function of our model that strongly penalizes a negative derivative of $f$. The monotonicity of the entire model (mean and standard deviation outputs) comes then from the monotonicity of the composition of monotonic functions. Notice that we are also imposing the standard deviation to be non-decreasing, assuming that the degradation of the model does not reduce the variability of our data.

The loss function to learn the trajectory performance library consists of three terms. The main term is in charge of learning the trajectory distributions of all training units, and the second and third terms enforce the monotonicity and smoothness of the latent function:

$$\sum_{i=1}^{N} \text{NLL}(h_i(t_1, t_2, \ldots, t_{L_i}) | p_1, p_2, \ldots, p_{L_i}) + \frac{\gamma_1}{M} \sum_{i=1}^{M} \max(f'(s_i), 0)^2 + \frac{\gamma_2}{M} \sum_{i=1}^{M} (f'''(s_i))^2,$$

where NLL denotes the negative log-likelihood of the normal distribution and $s_1 < s_2 < \cdots < s_M$ is a uniform partition of the minimal interval containing all the outputs of the training points through the input linear transformations $g_i^{\text{in}}$.

We also assume that the performance metric values are independent over time. Although performance values that are close in time should be correlated, we make this assumption for simplicity and to reduce the computational cost. An alternative is to model the covariance between different points in time with a kernel function like the radial basis function kernel used in Gaussian Processes. We plan to address this in future work.

## 2.3 RUL Prediction method

Our RUL prediction method is inspired by the model-based RUL prediction technique, where we first estimate the current state of the system, and then, using a transition model, we perform a Monte Carlo Simulation to predict the EOL of the system [10]. In our case, instead of a transition model for the system, we use training data of the performance metrics of the system. At a high level, our model estimates the current state for every performance metric by using the history of observations until the current time. It then predicts the EOL of each performance metric by combining this information with the library of performance distributions to predict their future evolution. This is done by performing an uncertainty propagation procedure to predict the set of possible ways the performance observation could evolve in time based on the observations and our library of trajectories learned from data. In this way, our method computes the possible EOL of each performance metric and the RUL of the system with uncertainty bounds.

### 2.3.1 Linear Space for Trajectory Searching

Our method is based on combining the present information that comes from the set of observations of the performance metric obtained until the current time step $n$ with the future information that comes from the library of trajectory distributions learned from data $\{h_1, h_2, \ldots, h_N\}$. We combine both by finding the linear combination of our trajectory library distributions that "best" fits our observations. Then, we consider the space of all possible trajectory distributions as the linear space generated by the set of distributions in our library

$$\mathcal{L}(h) = \left\{ \sum_{i=1}^{N} \alpha_i h_i : \alpha = (\alpha_1, \alpha_2, \ldots, \alpha_N) \in \mathbb{R}^N \right\}.$$

Note that this space contains many distributions that have no practical meaning. For example, if we restrict ourselves to the mean of the distributions, the constant function $g \equiv 0$ is always part of the space, independent of the library we have. Also, the linear combinations where all scalars are negative are defined in $\mathcal{L}(h)$, but this inverts the monotonicity of the resulting performance metric. We could use a more restrictive space that ensures more reasonable performance distributions like the space of convex combinations, also known as convex hull, of the (function) vectors $h = (h_1, h_2, \ldots, h_N)$, but this space is too restrictive for our purposes. We decided to keep our searchable trajectories as the linear space $\mathcal{L}(h)$ and avoid unrealistic trajectories through an intelligent search.

Notice that this is a linear space of Gaussian distributions. Since we trained our trajectory library to find the best Gaussian distribution fitting our training data, we know that $h_i = (\mu_i, \sigma_i)$ distributes $\mathcal{N}(\mu_i(t), \sigma_i^2(t)|t)$ and, under the common assumption of training trajectories independence, we obtain that the linear linear combination $\tilde{h} = \sum_{i=1}^{N} \alpha_i h_i \in \mathcal{L}(h)$ is also Gaussian with distribution

$$\mathcal{N}\left( \sum_{i=1}^{N} \alpha_i \mu_i(t), \sum_{i=1}^{N} \alpha_i^2 \sigma_i^2(t) \middle| \, t \right).$$

### 2.3.2 Finding the best Linear Combination

We formulate this problem by finding the best vector of scalars $\alpha$ that maximizes the $\alpha$-parameterized likelihood of the observations. As we mentioned when we learned the trajectory library, we also assume here independence over time for the same reasons.

Once we know the joint distribution of the linear combination of our library distributions (as a function of the scalars $\alpha$), we find the scalar values $\alpha$ of the linear combination that best fit the observations until the current time by maximizing the likelihood of the observation, or, equivalently, minimizing its negative log-likelihood.

In our problem, the parameters we are optimizing are the scalars of the linear combination $\alpha$. If we assume a Gaussian probability prior with mean $\bar{\alpha}$ and covariance matrix $\frac{1}{\gamma}I$, where $\gamma$ quantifies how much we believe that the right linear combination $\alpha$ is close to $\bar{\alpha}$,

$$\mathrm{NLL}_{\bar{\alpha},\gamma}(\alpha|\mathbf{p}_n) = \mathrm{NLL}(\alpha|\mathbf{p}_n) + \gamma(\alpha - \bar{\alpha})^T(\alpha - \bar{\alpha}) = \mathrm{NLL}(\alpha|\mathbf{p}_n) + \gamma\|\alpha - \bar{\alpha}\|^2, \tag{1}$$

where $\|\cdot\|$ denotes the euclidean norm in $\mathbb{R}^N$. Notice also that we have omitted the quantities independent of $\alpha$ as they do not affect the optimization.

### 2.3.3 Uncertainty Propagation

In model-based approaches, Monte Carlo simulations help to compute stochastically the possible ways in which the system may evolve under the influence of inputs and environmental conditions. Since we do not have a transition model, we cannot compute the next state given new inputs. Instead, we compute the most probable ways in which each performance metric may evolve, given the history of observations and a belief about its future evolution, by sampling the trajectories available in the library. Our proposed Uncertainty Propagation method is performed at each time step by solving the optimization problem with an objective function (1) for $\mathcal{N}(\bar{\alpha}, \frac{1}{\gamma}I)$ in a set of beliefs $\tilde{A}_n$ that change dynamically over time.

This belief set $\tilde{A}_n$ is composed of two belief sets, a present belief set $A_n^p$ in charge of tracking the best prediction at the present time, and a future belief set $A_n^f$, that keeps the library of trajectories as part of our belief space because sampling from them may help us propagate our predictions into the future. Therefore, our projections can accommodate and continue to be guided by future observations as they become available.

We first notice (see (1)) that the contribution of the observations until the present time, which we will call past information, and the present and future information generated by our belief sets should include the effect of the degradation and the environmental conditions at each time step $n$. At the beginning of the life of the system, when the degradation effects are small, the observations follow a mostly nominal behavior. Therefore, the prediction of the end of life derived from the performance metrics has large uncertainty bounds. Moreover, since we have a few observations of system performance and the time to system end-of-life is in the distant future, we have to rely more on the trajectories available in our library to predict the future performance evolution (therefore, $\gamma$ is larger). As the system approaches the end of life, we can give much larger weight to the set of observations on system behavior that we have accumulated because they contain much more information about degradation effects and the condition under which the system has operated. These are likely much better predictors of the system end of life than our prior beliefs of the system performance trajectories (implying $\gamma$ should be smaller). Assuming that the performance metric is decreasing and scaled in the interval $[0, 1]$, we can use the value of the current performance observation $p_n$ to weigh the past-future importance and balance these contributions in computing the EOL and RUL predictions. To reflect this, we modify (1) as follows

$$\text{objective}_{\mathbf{p}_n}(\alpha|\bar{\alpha}, \gamma) = (1 - p_n)\text{NLL}(\alpha|\mathbf{p}_n) + p_n\gamma\|\alpha - \tilde{\alpha}\|^2.$$

Note that, at the beginning of life, the observations reflect healthy performance values, so they are close to 1 and our optimization function mostly relies on our (present and future) beliefs. As the system degrades, the current performance observations, $p_n$, is used to give more weight to observations when performing the optimizations to compute the EOL.

In addition, we modify $\text{NLL}(\alpha|\mathbf{p}_n)$ to give more importance to fit recent performance observations than observations in time by doing a (linear) weighted sum

$$\text{NLL}(\alpha|\mathbf{p}_n) = \sum_{j=1}^{n} w_{j,n}\left[\frac{[\mathbf{p}_n - U\alpha]_j^2}{[V\alpha^2]_j} + \ln([V\alpha^2]_j)\right]$$

where $w_{i,n} = \text{Softmax}(w_{\max}(1 - p_i))$ for $i = 1, 2, \ldots, n$, $w_{\max}$ is a hyperparameter of the method indicating how much we care about future observation over past observations or the rate at which we *forget* past observations. At every time step $n$, given a set of beliefs $\tilde{A}_n = A_n^p \cup A_n^f$ and history of observations until current time $\mathbf{p} = (p_1, p_2, \ldots, p_n)$, we obtain the set of predictions of our uncertainty propagation method as

$$\tilde{A}_n^* = \left\{\arg\min_{\alpha\in\mathbb{R}^N}\{\text{objective}_{\mathbf{p}_n}(\alpha|\bar{\alpha}, \gamma)\} : \mathcal{N}\left(\bar{\alpha}, \frac{1}{\gamma}I\right) \in \tilde{A}_n\right\}.$$

If we obtain predictions that violate the monotonicity of the performance metric in the time interval $[t_n, t_{\max}]$, these predictions are pruned and we repeat the simulation for the number of pruned trajectories. This may happen because of the admission of negative values of the linear combination, so this prune, together with our belief (as we will explain later), enforces our search to be close to the convex hull of $h$.

### 2.3.4 Dynamic Beliefs

We already have discussed how to make performance predictions given observations and a belief set $\tilde{A}$, but how do we modify these sets over time? At each time step, the only information we need is the predictions we made in the previous step $A_{n-1}^{\text{pred}}$. Our method starts before we receive any observations ($n = 0$). At this initial state, we are already able to make predictions solely based on our trajectory library, as our Uncertainty Propagation method does when we have observations for $n \geq 1$. We make our first $n_{\text{paths}}$ predictions by doing $n_{\text{paths}}$ uniform sampling of the performance trajectories $h_i$, for $i = 1, 2, \ldots, N$. Then, at time step $n$, and given our previous predictions $A_{n-1}^{\text{pred}}$, we build our current predictions $A_n^{\text{pred}}$ inductively as follows:

**Correction Step.** Once we receive the predictions at the previous time step $A_{n-1}^{\text{pred}}$ we correct them in light of the new observation. This will give us the centers around which we will build our belief sets:

$$\alpha^{\text{present}} = \underset{\alpha \in A_{n-1}^{\text{pred}}}{\operatorname{argmin}} \{\text{NLL}(\alpha|\mathbf{p}_n)\}, \quad \alpha_i^{\text{future}} = \underset{\lambda \in \mathbb{R}}{\operatorname{argmin}} \{\text{NLL}(\lambda e_i|\mathbf{p}_n)\} e_i, \quad i = 1, 2, \ldots, N.$$

Notice that the center of our present belief is the prediction in $A_{n-1}^{\text{pred}}$ that has the smallest NLL of the observations(including the new observation $p_n$), while the $N$ centers of the future beliefs are the best scaling of the training trajectories fitting the observations. In order to assign a probability for each center to be selected, we also collect the minimum values of the future centers $q_i = \min_{\lambda \in \mathbb{R}} \{\text{NLL}(\lambda e_i|\mathbf{p}_n)\}$, and convert them into "probabilities" by applying the Softmax function to obtain the vector of probabilities $p^{\text{future}} = \text{Softmax}(q)$, where $q = (q_1, q_2, \ldots, q_N)^T$.

**Build the Belief sets.** Each belief in our belief sets is a Gaussian distribution $\mathcal{N}(\mu, \sigma^2 I)$. The means $\mu$ for both belief sets, $A_n^p$ and $A_n^f$, are constructed by a Gaussian sampling around their centers. The sampling here simulates our selection of different beliefs around a center and has no probabilistic implications in our model other than an automatic and "reasonable" way to select different beliefs. On the other hand, the variance $\sigma^2$ only depends on the centers. All the beliefs sampled around the same center share the same variance, $\sigma^2$. We refer to the inverse of the variance of a center as the trust in the center, denoted by $\gamma$. Since we have one present center and $N$ future centers, we denote their respective trusts by $\gamma^p$ for the present center and $\gamma_i^f$ for the $i$-th future center.

Since our beliefs are on trajectories and we only have access to the scalars that generate these trajectories, adding a Gaussian noise directly in the scalars creates a bias towards the origin of coordinates. But there is no reason to have a stronger belief about the constant zero trajectory, so we make a change of base from $h$ to the mean-centered base $\{h_i - \bar{h}\}_{i=1}^N$, where $\bar{h} = \frac{1}{n}\sum_{i=1}^N h_i$, before applying the Gaussian noise on the scalars. This way we move the origin of coordinates to the mean trajectory $\bar{h}$.

At this point, we only need to determine the vector of variances of the samples $V^{\text{present}}$ ,$V_{i,n}^{\text{future}}$, for $i = 1, 2, \ldots, N$, the center trusts $\gamma^p$ and $\gamma_i^f$, and how the $n_{\text{paths}}$ samples, at time $n$, are distributed across the past and the future belief sets. We compute the variance of the sample belief as how far (at each coordinate) the center is to be the mean of our predictions $A_{n-1}^{\text{present}}$ or equivalently, "how big" was the correction we had to make in the previous step. We quantify this by computing the variance of the predictions around each center $V_n^{\text{present}} = V(A_{n-1}^{\text{pred}}, \alpha^{\text{present}})$ and $V_{i,n}^{\text{future}} = V(A_{n-1}^{\text{pred}}, \alpha_i^{\text{future}})$, for $i =, 1, 2, \ldots, N$, where the variance of a set $\{x_n\}_{i=1}^m$ centered at a vector $x$ is almost the same as the variance of the set $\{x_n\}_{i=1}^m$ with $x$ in place of the mean in the formula for the empirical variance. This means $V(\{x_n\}_{i=1}^m, x) = \frac{1}{m-1}\sum_{i=1}^m (x_i - x)^2$.

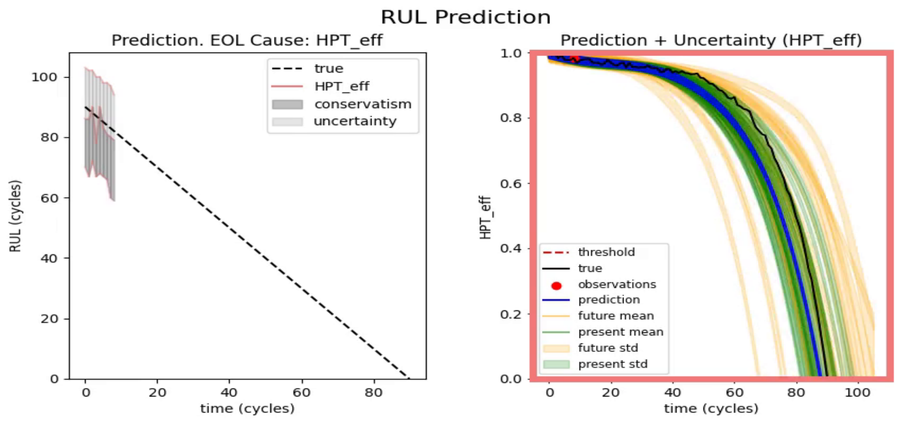


**Figure 3** RUL prediction (on the left) and Performance prediction on the right on Unit 7 of N-CMAPSS-DS01 dataset. The green and yellow predicted distributions correspond to resulting distributions after the uncertainty propagation process with prior belief in $A_n^p$ and $A_n^f$ respectively. Tee blue distribution is the mean predicted distribution, the one we use for the RUL prediction.

We consider our trust in the respective beliefs as the inverse of the mean along components of the vector variance. This avoids the non-intuitive idea of dividing the belief by components of the linear combination. Remember our belief is on trajectories, and during the conversion from scalars to trajectories, many unrealistic trajectories may appear by just changing the sign of scalars. We also checked in practice that this gives us more stable predictions as variations in all dimensions are equal. Then, we define the center trusts as

$$\gamma_n^{\text{present}} = \frac{1}{\frac{1}{N}\sum_{j=1}^{N}[V_n^{\text{present}}]_j}, \qquad \gamma_{i,n}^{\text{future}} = \frac{1}{\frac{1}{N}\sum_{j=1}^{N}[V_{i,n}^{\text{future}}]_j}, \quad i = 1, 2, \ldots, N.$$

Finally, we chose the (present-future) sample rate as

$$r_n = \frac{\gamma_n^{\text{present}}}{\gamma_n^{\text{present}} + \frac{1}{N}\sum_{i=1}^{N}\gamma_{i,n}^{\text{future}}} \in [0,1].$$

to give more beliefs to the centers we trust more. So, at time step $n$, the belief distribution is $|A_n^p| = \lfloor r_n n_{\text{paths}} \rfloor$ and $|A_n^f| = \lceil (1-r_n) n_{\text{paths}} \rceil$ where $\lfloor x \rfloor$ denotes the floor function, $\lceil x \rceil$ denotes the ceiling function, and $|A|$ denotes the cardinality of the set $A$.

**Uncertainty Propagation.** Once we have built our belief set, we perform our Uncertainty Propagation step explained in the previous section, for the entire belief set. The $\alpha^*$ values obtained are the predictions at time step $n$. This means $A_n^{\text{pred}} = \tilde{A}_n^* = (A_n^{\text{p}})^* \cup (A_n^{\text{f}})^*$. Figure 3 on the right, shows the performance prediction obtained from the scalars in the prediction set $A_n^{\text{pred}}$, at a specific time step, differentiating the future and present belief predictions by color.

### 2.3.5 RUL and Uncertainty Computation

At each time $t_n$ (starting by $t_0 = 0$) and for each of the $\kappa$ performance metrics, we compute the prediction sets $A_{j,n}^{\text{pred}}$, for $j = 1, 2, \ldots, \kappa$, following the steps explained in Section 2.3.4 and we use these scalar sets to compute the set of $n_{\text{paths}}$ prediction distributions

$$\text{Pred}_j(t_n) = \left\{ \sum_{i=1}^{n_{\text{paths}}} \alpha_i h_i : \alpha \in A_{j,n}^{\text{pred}} \right\}, \text{ for } j = 1, 2, \ldots, \kappa.$$

We obtain the predicted $j$-th performance metric at time step $n$, $\mathbf{f}_{j,n}$, by taking the average in both the mean and standard deviation of the predicted distributions $\text{Pred}_j(t_n)$. This means $\mathbf{f}_{j,n} \sim \mathcal{N}\left(\mu_{j,n}(t), \sigma^2_{j,n}(t)|t\right)$, where

$$(\mu_{j,n}(t), \sigma_{j,n}(t)) = \frac{1}{n_{\text{paths}}} \sum_{\mathcal{N}(\mu,\sigma)\in \text{Pred}_j(t_n)} (\mu(t), \sigma(t)).$$

Notice that we are taking the Gaussian distribution whose mean and standard deviation are the average of the mean and standard deviation of the predicted distributions, which is different from the average distributions, which would reduce the standard deviation by a factor of $\frac{1}{\sqrt{n_{\text{paths}}}}$. Assuming that the performance is decreasing, we compute the predicted EOL of the performance $j$ at time $t_n$ and its uncertainty boundaries as

$$\begin{aligned}
\text{EOL}_j(t_n) &= \min\{t \in [t_n, t_{\max}] : \mu_{j,n}(t) - 1.96\sigma_{j,n}(t) \le 0\},\\
\text{EOL}_j^{\text{low}}(t_n) &= \min\{t \in [t_n, t_{\max}] : \mu(t) - 1.96\sigma(t) \le 0, \mathcal{N}(\mu,\sigma^2) \in \text{Pred}_j(t_n)\},\\
\text{EOL}_j^{\text{up}}(t_n) &= \max\{t \in [t_n, t_{\max}] : \mu(t) + 1.96\sigma(t) \ge 0, \mathcal{N}(\mu,\sigma^2) \in \text{Pred}_j(t_n)\}.
\end{aligned}$$

Note that we are being conservative at choosing the lower bound of the 95% confidence interval $[\mu_{j,n}(t) - 1.96\sigma(t), \mu_{j,n}(t) + 1.96\sigma(t)]$. Finally, we predict the EOL and RUL of the system with its respective confident bounds at time $t_n$ as follows

$$\begin{aligned}
\text{EOL}(t_n) &= \min_{j=1,2,\ldots,\kappa}\{\text{EOL}_j(t_n)\}, & \text{RUL}(t_n) &= \text{EOL}(t_n) - t_n,\\
\text{EOL}^{\text{low}}(t_n) &= \min_{j=1,2,\ldots,\kappa}\{\text{EOL}_j^{\text{low}}(t_n)\}, & \text{RUL}^{\text{low}}(t_n) &= \text{EOL}^{\text{low}}(t_n) - t_n,\\
\text{EOL}^{\text{up}}(t_n) &= \min_{j=1,2,\ldots,\kappa}\{\text{EOL}_j^{\text{up}}(t_n)\}, & \text{RUL}^{\text{up}}(t_n) &= \text{EOL}^{\text{up}}(t_n) - t_n.
\end{aligned}$$

## 3 Results

### 3.1 Dataset

We conducted experiments to validate our method on the N-CMAPSS dataset introduced in [1]. We used datasets 1 (N-CMAPSS-DS01) and 5 (N-CMAPSS-DS05). We selected N-CMAPSS-DS01 because it only had one failure mode and N-CMAPSS-DS05 as a more complex dataset with two failure modes. The N-CMAPSS dataset does not provide the performance metrics used to evaluate the health of the system, but following the instructions given in the paper, we can compute them by normalizing the health parameters with respect to the threshold using the formula $p(t) = 1 - \frac{\mu(t)-\mu_{\text{new}}}{\text{thr}}$, where $\mu$ denotes the health parameter, $\mu_{\text{new}}$ is the nominal value of the degradation parameter (with no degradation) and thr is the threshold. We approximate these two operating margins by taking the maximum and minimum along all the health parameter values of all the training units. In the case of N-CMAPSS-DS01, only the efficiency of the high-pressure turbine (HPT eff) is affecting the degradation of the system, while in dataset N-CMAPSS-DS05 we have the flow and efficiency high-pressure compressor (HPC flow and HPC eff). We selected one sample per cycle as a sample rate for training and testing.

### 3.2 MTL on N-CMAPSS dataset

We start by training our MTL model explained in Section 2.2 to learn the trajectory libraries (one for each performance metric). Each model was trained 10 times for each performance metric, changing weight initialization to account for model uncertainty. This is important because, although all models should agree during the training period, the model uncertainty should increase as we learn beyond the training time. So, training several models and building an ensemble of them gives us a good estimation of the model uncertainty in the prolongation.

## 3.3 RUL prediction on the N-CMAPSS dataset

We evaluated our method on the four testing units in both datasets. To evaluate and compare our results, we used four evaluation metrics. Two for evaluating the prediction and the other two for the uncertainty bounds. To evaluate the prognostics' results, two commonly used metrics in prognostics analysis are the root-mean-square error (RMSE) and NASA's scoring function [16].

The latter metric penalizes over-predicting as this may cause serious damage to the system unlike an under-prediction error, which may only cause unnecessary extra maintenance of the system. To quantify the uncertainty on the RUL predictions, two widely used metrics are the Prediction Interval Coverage Probability (PICP) which quantifies the probability that the true RUL falls inside the corresponding prediction interval, and the prediction interval normalized width (PINAW), which represents the width of the prediction interval normalized on the range of actual RUL values.

We applied our method by taking the hyperparameters: $n_{\text{paths}} = 60$ and forgetting coefficient $w_{\max} = 1$, which implies almost no forgetting. Tables 1 summarize the results of our model (**LibRUL**) on both dataset N-CMAPSS-DS01 and N-CMAPSS-DS05, compared with some state-of-the-art uncertainty prediction models. The other model results have been taken from Tables 4 and 6 in [21]. The worst and best RUL prediction results on units of this dataset are shown in Figures 4a and 4b.

**Table 1** Results for RUL prediction with uncertainty quantification in N-CMAPSS-DS01 and N-CMAPSS-DS05. Ours is LibRUL. Baseline results taken from [21].

| | N-CMAPSS-DS01 | | | | N-CMAPSS-DS05 | | | |
|---|---|---|---|---|---|---|---|---|
| **Model** | **RMSE** | **Score** | **PICP** | **PINAW** | **RMSE** | **Score** | **PICP** | **PINAW** |
| LibRUL | 6.551 | 250.392 | 81.82% | **23.01%** | 6.077 | 184.166 | **91.74%** | 29.08% |
| BGT | **4.285** | 131.697 | **96.77%** | 27.16% | 6.218 | 192.52 | 87.77% | **27.14%** |
| BLSTM | 4.849 | 137.029 | N/A | N/A | 6.915 | 278.689 | N/A | N/A |
| LSTM | 5.396 | 195.57 | N/A | N/A | 7.522 | 263.393 | N/A | N/A |
| DAT | 4.57 | **74** | N/A | N/A | **5.18** | **99** | N/A | N/A |
| DGP | 7.03 | 14213.9 | N/A | N/A | 8 | 15499.2 | N/A | N/A |
| GT | 4.823 | 149.3 | N/A | N/A | 6.413 | 251.25 | N/A | N/A |

The results on the N-CMAPSS-DS01 dataset are slightly behind the state-of-the-art solutions. This is because two of the testing units, specifically units 7 and, especially, unit 9, fall outside the linear space $\mathcal{L}(h)$ generated by the library. These units exhibit a more accelerated degradation than any trajectory in the training dataset. To accurately capture this behavior, we would need to blend the training curves, modifying their curvature (through second-order transformations) to match these shapes. However, this level of curvature cannot be achieved through linear combinations of the library. The primary limitation of our approach is its inability to represent such curvature using linear combinations, which is a priority for future work and will be our immediate next step.

On the other hand, for N-CMAPSS-DS05, our model outperforms almost all state-of-the-art models (except for DAT) in both RMSE and Score, as shown in Table 1. This is expected, as DAT is a domain adaptation approach that uses significantly more data. Additionally, uncertainty quantification is not a focus of the DAT method. DAT employs Bayesian deep learning techniques, specifically dropout layers, which provide only limited quantification of prediction uncertainty. Meanwhile, the BGT model is approximately 2% better than our method in terms of the uncertainty bound width of the predictions, but this comes at the cost of a 4% reduction in prediction interval coverage.

These results highlight the strengths of our model when the testing performance trajectories are "similar" to the training trajectories, especially when considering the increased explainability of our approach. Both BGT and DAT, while admitting uncertainty estimation, largely operate as "black-box" models. Although they effectively capture degradation patterns, they do not provide clear explanations for why specific predictions are made.

It is also important to note that our predictions are based solely on performance metrics, without leveraging feature information that could provide additional degradation insights. We also used a sampling rate of one cycle, which is equal to or greater than that of all competing approaches.

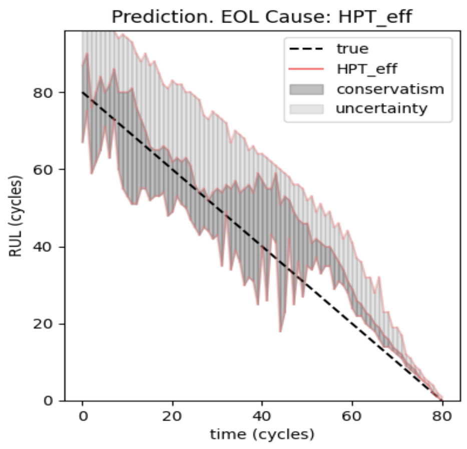
The worst and best RUL prediction results for units in this dataset are shown in Figures 4c and 4d. Note also that our predictions become very precise, with significantly less uncertainty toward the end of the lifecycle compared to state-of-the-art results. This is because our method balances present and future uncertainty as more observations are collected, and in the final stages, future uncertainty is significantly reduced. This is paramount in prognostics to avoid system failure. The only exception is in Figure 4c, where a precision delay occurs because one of the two performance metrics that drive the RUL estimation process remains very healthy, far from showing a clear degradation pattern and resulting in a larger uncertainty while the other performance metric suddenly decreases in performance.

An additional feature of our model is that it also predicts the cause of failure at each time step given by the performance that is predicted to cross the threshold earlier. Figures 4c and 4d show this in the color of the RUL predictions. Notice that in the case of 4c, the cause of failure is predicted correctly from a very early time step (around cycle 30).

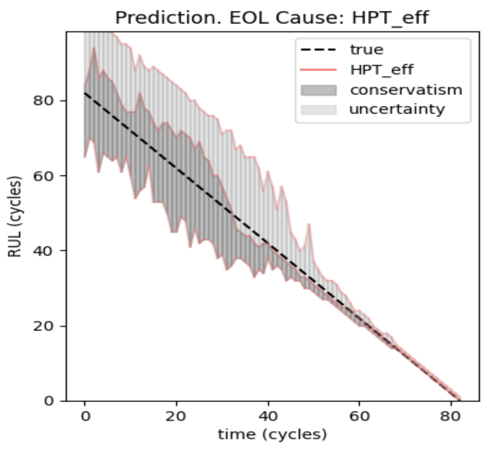
## 4 Conclusions

This paper presents a novel data-driven methodology for prognostics, inspired by the model-based RUL estimation approach. Our proposed approach bridges the gap between these two paradigms by developing a data-driven method that mimics the reasoning processes of model-based approaches. It leverages performance metrics and their associated thresholds to predict RUL in real-time, without requiring RUL labels, which are expensive and labor-intensive to acquire. Additionally, our model quantifies uncertainty in its predictions, offering valuable insights into which specific performance metric thresholds have been violated. This feature enhances the explainability of the RUL prediction process, making it more transparent and actionable for decision-makers.

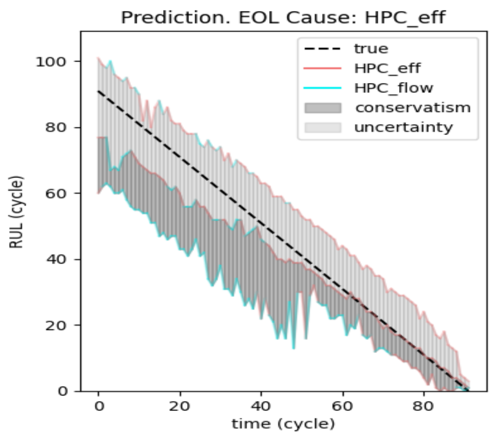
We empirically validated our approach using two datasets from the widely used N-CMAPSS benchmark. The results demonstrate the effectiveness of our methodology in predicting RUL while also providing confidence bounds for the predictions. For one of the datasets, the results are very close to the best state-of-the-art performance. However, in the other dataset, our results fall behind the current state of the art, highlighting a limitation of our approach when testing trajectories are not linearly representable by the training data. Our model struggles with datasets where the test data distribution significantly diverges from the training data. Nevertheless, the combination of explainability, uncertainty quantification, and online prediction makes our approach a promising alternative to both purely data-driven and traditional model-based methods, particularly in the context of complex systems. Future



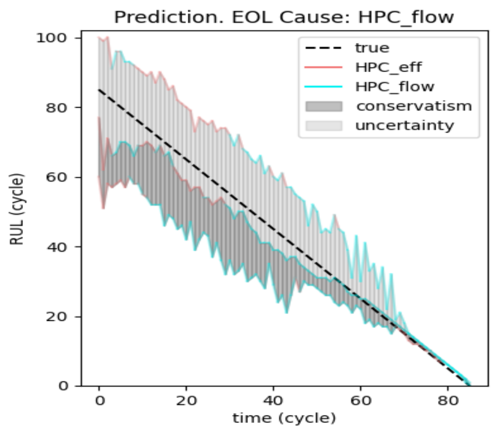
**(a)** RUL predictions on the C-MAPSS-DS01. Unit 9.



**(b)** RUL predictions on the C-MAPSS-DS01 Unit 10.



**(c)** RUL predictions on the C-MAPSS-DS05. Unit 8.



**(d)** RUL predictions on the C-MAPSS-DS05. Unit 7.

**Figure 4** RUL predictions examples. Worst example on the left, best example on the right for each dataset. The color in the predictions indicates the performance that, at current time, the model predicts is going to cross the threshold first. The line in the middle represents the predicted value.

work will explore addressing the out-of-distribution issue by augmenting the model's ability to generalize beyond simple linear combinations of training data. Specifically, we are working on approaches to augment the performance model's predictions accounting for past data without constraining the model's inference space to a linear combination of past trajectories. This is especially important when predicting for degrading conditions which have not been experienced in the past, i.e. there is not representative data available for these operating conditions.

## References

1 Manuel Arias Chao, Chetan Kulkarni, Kai Goebel, and Olga Fink. Aircraft engine run-to-failure dataset under real flight conditions for prognostics and diagnostics. *Data*, 6(1):5, 2021. doi:10.3390/DATA6010005.

2 Piero Baraldi, Francesca Mangili, and Enrico Zio. A prognostics approach to nuclear component degradation modeling based on gaussian process regression. *Progress in Nuclear Energy*, 78:141–154, 2015.

3 Yuanhong Chang, Fudong Li, Jinglong Chen, Yulang Liu, and Zipeng Li. Efficient temporal flow transformer accompanied with multi-head probsparse self-attention mechanism for remaining useful life prognostics. *Reliability Engineering & System Safety*, 226:108701, 2022. doi:10.1016/J.RESS.2022.108701.

4 Manuel Chao, Chetan Kulkarni, Kai Goebel, and Olga Fink. Fusing physics-based and deep learning models for prognostics. *Reliability Engineering & System Safety*, 217:107961, 2022. doi:10.1016/J.RESS.2021.107961.

5 Matthew Daigle and Kai Goebel. Model-based prognostics with concurrent damage progression processes. *IEEE Transactions on Systems, man, and cybernetics*, 43(3):535–546, 2012. doi:10.1109/TSMCA.2012.2207109.

6 Matthew Daigle, Bhaskar Saha, and Kai Goebel. A comparison of filter-based approaches for model-based prognostics. In *IEEE Aerospace conference*, pages 1–10, 2012.

7 Austin Downey, Yu-Hui Lui, Chao Hu, Simon Laflamme, and Shan Hu. Physics-based prognostics of lithium-ion battery using non-linear least squares with dynamic bounds. *Reliability Engineering & System Safety*, 182:1–12, 2019. doi:10.1016/J.RESS.2018.09.018.

8 Jian Guo, Zhaojun Li, and Meiyan Li. A review on prognostics methods for engineering systems. *IEEE Transactions on Reliability*, 69(3):1110–1129, 2019. doi:10.1109/TR.2019.2957965.

9 Cheng-Geng Huang, Hong-Zhong Huang, and Yan-Feng Li. A bidirectional lstm prognostics method under multiple operational conditions. *IEEE Transactions on Industrial Electronics*, 66(11):8792–8802, 2019. doi:10.1109/TIE.2019.2891463.

10 Hamed Khorasgani, Gautam Biswas, and Shankar Sankararaman. Methodologies for system-level rul prediction. *Reliability Engineering & System Safety*, 154:8–18, 2016.

11 Jianfeng Lv, Zhongliang Yu, Haochuan Zhang, Guanghui Sun, Patrick Muhl, and Jianxing Liu. Transformer based long-term prognostics for dynamic operating pem fuel cells. *IEEE Transactions on Transportation Electrification*, 10(1):1747–1757, 2023.

12 Mohamed Marei and Weidong Li. Cutting tool prognostics enabled by hybrid cnn-lstm with transfer learning. *The International Journal of Advanced Manufacturing Technology*, 118(3):817–836, 2022.

13 Khanh TP Nguyen, Kamal Medjaher, and Christian Gogu. Probabilistic deep learning methodology for uncertainty quantification of remaining useful lifetime of multi-component systems. *Reliability Engineering & System Safety*, 222:108383, 2022. doi:10.1016/J.RESS.2022.108383.

14 Seol-Hyun Noh. Analysis of gradient vanishing of rnns and performance comparison. *Information*, 12(11):442, 2021. doi:10.3390/INFO12110442.

15 Darius V Roman, Ross W Dickie, David Flynn, and Valentin Robu. A review of the role of prognostics in predicting the remaining useful life of assets. In *27th European Safety and Reliability Conference 2017*, pages 897–904. CRC Press, 2017.

16 Abhinav Saxena, Kai Goebel, Don Simon, and Neil Eklund. Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 international conference on prognostics and health management*, pages 1–9. IEEE, 2008.

17 Zhigang Tian. Artificial neural network method for remaining useful life prediction of equipment subject to condition monitoring. *Journal of intelligent Manufacturing*, 23:227–237, 2012. doi:10.1007/S10845-009-0356-9.

18 A Vaswani. Attention is all you need. *Advances in NeurIPS*, 2017.

19 Simon Vollert and Andreas Theissler. Challenges of machine learning-based rul prognosis: A review on nasa's c-mapss data set. In *2021 26th IEEE international conference on emerging technologies and factory automation (ETFA)*, pages 1–8. IEEE, 2021. doi:10.1109/ETFA45728.2021.9613682.

20 Ping Wang, Zhiqiang Long, and Gao Wang. A hybrid prognostics approach for estimating remaining useful life of wind turbine bearings. *Energy Reports*, 6:173–182, 2020.

21 Feifan Xiang, Yiming Zhang, Shuyou Zhang, Zili Wang, Lemiao Qiu, and Joo-Ho Choi. Bayesian gated-transformer model for risk-aware prediction of aero-engine remaining useful life. *Expert Systems with Applications*, 238:121859, 2024. doi:10.1016/J.ESWA.2023.121859.