Robust Remaining Useful Life Prediction Using Jacobian Feature Regression-Based Model Adaptation

Prasham Sheth[1] and Indranil Roychoudhury[1]

[1] *SLB, Menlo Park, California, 94025, USA*
psheth2@slb.com
iroychoudhury@slb.com

Abstract

The accurate and robust prediction of remaining useful life (RUL) is critical for enabling the proactive mitigation of fault effects rather than reacting to them. For RUL prediction, one must model nominal and faulty system behaviors and how different faults progress over time. Complex data-driven machine learning (ML) models may capture both nominal and fault progression by updating the model parameters at different stages. As new data are observed, these model parameters can be updated to keep the system model always accurate. However, complete retraining of these models is both data- and computation-intensive and unsuitable for dynamic, fast-changing environments requiring quick recalibration. This calls for efficiently adapting the model to new operating conditions or the system's current state. One such efficient way to recalibrate model parameters to newly observed data using Jacobian feature regression (JFR) is presented in Forgione, Muni, Piga, and Gallieri (2023), where a recurrent neural network (RNN) models the current behavior of the dynamic system. Then, any subsequent deviation of observed measurements and the RNN model is attributed to an "unacceptable degradation of the nominal model performance." To update the RNN model, Forgione et al. (2023) propose augmenting the current model with additive correction terms learned by implementing JFR on observed "perturbed system" data. In this paper, we propose an automated *online* framework to adapt the model efficiently to always reflect the system's current state and use it for accurate RUL prediction and select JFR as one such adaptation technique. We extend the implementation of JFR-based model adaptation to hybrid models and demonstrate JFR to be more sustainable than the other retraining methods. Finally, we showcase the application of this approach to the oil and gas industry. A testbed that simulates a digital synthetic oilfield is used to show the effectiveness of this adaptation-based RUL prediction technique.

1. Introduction

1.1. Motivation

The accurate and robust prediction of remaining useful life (RUL) is one of the most critical functions of prognostics and health monitoring for assets. RUL is the time before a system can no longer function nominally. Knowing how much useful time a system has can help prevent having to react to such a failure and plan steps to mitigate its effects, e.g., planning maintenance and repair, thereby supporting the seamless operation of the facility.

Predicting RUL is a well-explored problem and can be implemented using (1) *model-based* or science-based approaches, (2) *data-driven* approaches that use available data but completely agnostic of scientific domain knowledge, or (3) *hybrid* approaches that combine scientific theory with available data. For accurate RUL prediction, a model must capture the nominal behavior of the system as well as the progressively degraded behavior. In model-based approaches, multiple models can be developed for nominal and degraded system behavior for different possible faults. A single, sufficiently complex data-driven or hybrid model may capture both nominal and degraded system behavior. One approach to building a single model is to periodically update or adapt the model to new system observations so as to reflect reality accurately. Many possible ways are available in the literature that enable this updating or adaptation of the model, such as retraining of machine learning (ML) models, recalibration of model parameters based on some initially collected field data, and so on. However, many of these approaches are computationally expensive, have intense data requirements, and are unsuitable for dynamic, fast-changing environments that need quick recalibration requirements.

1.2. Related Work and Their Challenges

Wang, Zhao, and Addepalli (2020) provide a well-structured summary of the increasing interest in using deep-learning approaches, such as autoencoders, deep belief networks, recurrent neural networks (RNNs), and convolutional neural networks for RUL prediction. The authors present conventional model-based as well as hybrid RUL prediction approaches and provide an excellent summary of the scope of the research development undertaken regarding RUL prediction.

To highlight a few works that focus on predicting RUL, Lei et al. (2016) propose an approach that contains two modules: (1) indicator construction, which fuses the information from multiple features and constructs a health indicator that they referred to as weighted minimum quantization error to correlate the machinery degradation, and (2) RUL prediction block that uses a particle-filtering-based algorithm. Ma and Mao (2021) propose a convolution-based long short-term memory (CLSTM) for predicting the RUL of bearings. By showcasing the ability of the CLSTM architecture to predict RUL effectively, they validate the effectiveness of using the spatial and temporal features. Y. Zhang, Xiong, He, and Liu (2017) showcase the use of an LSTM-RNN based method for predicting the RUL of lithium-ion batteries, highlighting the approach's capability to capture the long-term dependencies of capacity degradation in batteries.

A major challenge of the existing RUL prediction approaches is that they are system-state-dependent; i.e., their effectiveness in predicting RUL accurately is hinged on the assumption that the system model accurately reflects the current and future (degraded) system states. However, collecting data representing the system in all possible scenarios is impossible for many systems in real life, especially when these systems are never allowed to degrade sufficiently. The data collection process for such a wide range of possibilities is also extremely expensive. Therefore, the availability of training data for such models is a big bottleneck, and as soon as the system goes into a level of degradation that is not represented in the data, the chances of the model effectively predicting RUL degrades exponentially. In such scenarios, having a dynamic model that adapts based on the system's state becomes necessary. Researchers have focused on developing different approaches by using techniques such as transfer learning, domain adaptation, and modeling the degradation process. This does not represent the exhaustive list of techniques but just highlights the major approaches being explored.

Transfer learning is one technique for adapting the model to use the previous learnings to improve generalization about the new task. In the case of the RUL prediction, the task remains the same but differs as the underlying data distribution changes; hence, the model has to be "transferred" to this new set of data points that belong to the new distribution. The change in distribution, as mentioned earlier, could result from different operating conditions or different states of the system because of the degradation or upgrades to the equipment of the system. *Domain adaptation* falls within the umbrella of transfer learning, wherein the main focus is to help the model adapt from one or more sources of domains to a target domain. Ding, Ding, Zhao, Cao, and Jia (2022) introduce a multisource domain adaptation network for RUL prediction of bearing under varying conditions. Their domain adaptation strategy functions in two stages where the domain-specific distribution is integrated with regressor adaptation. A fusion of LSTM and domain adversarial neural networks (DANN) to extract temporal information from the time-series data and learn the domain-invariant features, thereby successfully addressing the challenge of distribution shifts in data domains resulting from the different states of the systems is proposed in da Costa, Akçay, Zhang, and Kaymak (2020). Si, Hu, Chen, and Wang (2011) present an approach to predict RUL using a Wiener process with a nonlinear and time-dependant drift coefficient. It, in particular, involves designing a state-space model and using Bayesian filtering to update the drifting function parameter. The method is significant because of its potential application in online prediction, which is one of the critical requirements for such an RUL prediction framework. With different dynamics of the underlying system, determining the frequency of the updates is a critical task. L. Liu, Guo, Liu, and Peng (2019) introduce a data-driven framework for RUL prediction that integrates sensory anomaly detection and data recovery and improve RUL prediction by detecting sensory anomalies, recovering data, and using this recovered data for more accurate predictions. Huang, Xu, Wang, and Sun (2015) focus on addressing nonlinear degradation trajectories and heterogeneity in practical systems. They combine a nonlinear Wiener process with an adaptive drift feature. Y. Zhang, Yang, Xiu, Li, and Liu (2021) present an integrated technique that combines the Wiener process for degradation modeling and an LSTM network for forecasting degradation increments by learning the long-term dependencies of the offline degradation model and online observed degradation. Cheng et al. (2023) focus on predicting the RUL of the machinery under varying working conditions. Their proposed approach uses dynamic domain adaptation by integrating dynamic distribution and adversarial adaptation networks to predict RUL effectively. Pan, Li, and Wang (2022) propose combining LSTM and particle filter to predict the RUL for lithium-ion batteries under different stress conditions. They particularly leveraged transfer learning to update the LSTM model to ensure generalizability and then particle filter to capture the uncertainty. Siahpour, Li, and Lee (2022) introduce consistency-based regularization into the DANN training process. Consistency-based regularization helps to remove the negative impact of missing information. Sun et al. (2019) present a deep transfer learning network based on spare autoencoder. They incorporate three strategies — weight transfer, feature transfer learning, and weight update — to improve adaptability and prediction accuracy. Also, they showcased the network's capability to be trained on one tool and then being transferred to another tool under operation for online RUL prediction. The use of bi-directional LSTM (BLSTM) neural networks and transfer learning for RUL estimation is explored in A. Zhang et al. (2018). They address the challenges of insufficient failure progression samples in data-driven prognostics by training the model on different but related datasets and then fine-tuning it with the real target domain dataset. J. Liu, Saxena, Goebel, Saha, and Wang (2010) present an adaptive recurrent neural network (ARNN) model for predicting the RUL of lithium-ion batteries. The model employs a dynamic state forecasting approach using a neural network architecture that adapts by optimizing the model's weights using the recursive Levenberg-Marquardt method.

1.3. Proposed Solution and Contributions

A better way to recalibrate model parameters to match model predictions to the newly observed data using Jacobian feature regression (JFR) is presented in Forgione et al. (2023). An RNN is used to model the dynamic system using available measurements. Then, as the system dynamics change, it causes the nominal model to be inaccurate for predicting the observed measurements in the presence of the perturbed system dynamics. The core idea of their approach is to adapt an existing RNN model, which was trained on data from a nominal system, to perturbed system dynamics, not by retraining the model from scratch, but by including an additive correction term to the nominal model's output. This correction term is designed to account for the discrepancies between the nominal system and the perturbed system. In other words, as an "unacceptable degradation of the nominal model performance" occurs, Forgione et al. (2023) propose a transfer learning approach to improve the performance of the nominal model in the presence of perturbed system dynamics, where the nominal model is augmented with additive correction terms that are trained on observed perturbed system data. These correction terms are learned through JFR "defined in terms of the features spanned by the model's Jacobian concerning its nominal parameters." Efficient model adaptation is achieved by using the JFR in the feature space defined by the Jacobian of the model with respect to its nominal parameters. Forgione et al. (2023) also propose a non-parametric view that uses the Gaussian process. This could be useful to provide flexibility and efficiency for very large networks or when only a few data points are available.

The contributions of this work are significant because they offer a more efficient and effective way to keep data-driven and hybrid models accurate when applied to dynamical systems that experience changes over time. We address some of the challenges described in Section 1.2 by building upon the method introduced in Forgione et al. (2023) as follows:

1. We present an automated approach to use adaptation techniques for predicting RUL while ensuring system-state-dependency of the models. Although JFR is used as the model-adaptation technique in this paper, JFR can be replaced by any other model-adaptation algorithm without any loss of generalizability.
2. We extend the implementation of JFR-based model adaptation to hybrid models that combine physics and data-driven models. This is important (and even necessary) as the representation of systems using data-driven models can become a bottleneck if the training data are limited, and there could be no guarantee that the data-driven models follows the physics of the system behaviors in all possible scenarios.
3. We highlight the lower carbon footprint of the JFR-based adaptation technique instead of retraining the model completely using the standard transfer learning.
4. We modify the offline adaptation approach into an online adaptation approach, which becomes critical to the PHM systems, especially in RUL prediction. To enable online adaptation, we use the anomaly detection output in order to trigger the model-adaptation.
5. Finally, we also discuss the application of our JFR-based model adaptation approach to assets relevant to the oil and gas industry. In particular, we discuss the results of applying the technique to a testbed that simulates a digital synthetic oilfield.

1.4. Organization

The remainder of this paper is organized as follows. Section 2 formulates the RUL prediction problem, and our approach to solve this. Section 3 includes the experimental setup and results, and finally, Section 4 concludes the paper and provides directions for future work.

2. Problem Setup and Approach

To set up the problem, let us denote a system by S, that typically takes in some inputs from discrete timesteps 1 to k, i.e., $\mathbf{x}_{1:k}$, and has measured outputs $\mathbf{y}_{1:k}$. Let us assume that M denotes a model representing this system S that takes in the same inputs $\mathbf{x}_{1:k}$ and outputs simulated measurements denoted by $\hat{\mathbf{y}}_{1:k}$. Due to differences between a model and reality, such as modeling error, and measurement noise, $\mathbf{y}_{1:k}$ and $\hat{\mathbf{y}}_{1:k}$ are seldom exactly the same, but a "good" model M would generate $\hat{\mathbf{y}}_{1:k}$ that is very close to the real outputs $\mathbf{y}_{1:k}$. We define a *threshold function* $T : \hat{\mathbf{y}}_{\kappa} \rightarrow \{\texttt{true}, \texttt{false}\}$ that partitions the operational state of the system into nonfailure and failure states based on observed measurements, such that $T(\mathbf{y}_{\kappa})$ returns `true` when the system is in a failure state, and `false` otherwise. If the time of prediction is denoted by k_P, then typically, we define end-of-life (EOL) predicted at time k_p as $EOL(k_P) = \inf\{k' : k' \geq k_P \text{ and } T(\hat{\mathbf{y}}_k)\}$, and RUL at time k_P is defined as $RUL(k_P) = EOL(k_P) - k_P$. To make EOL and RUL predictions, the model M is fed hypothesized future inputs $\mathbf{x}_{k_P:\infty}$, also denoted by $\mathbf{x}_{future}$.

The model M can be defined/trained using model-based, data-

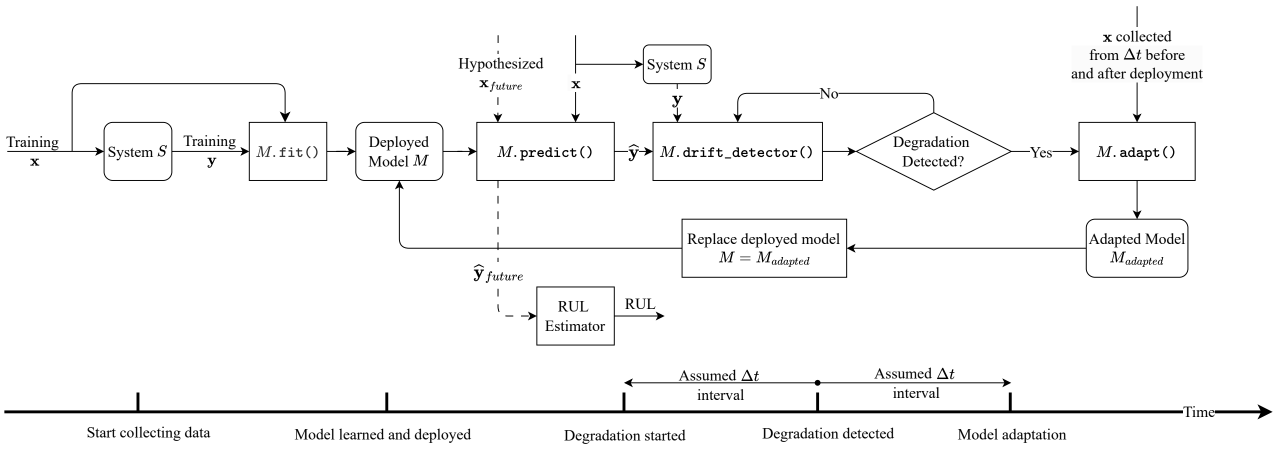

Training $\mathbf{x}$
System S
Training $\mathbf{y}$
$M.\texttt{fit()}$
Deployed Model M
Hypothesized $\mathbf{x}_{future}$
$\mathbf{x}$
System S
$M.\texttt{predict()}$
$\hat{\mathbf{y}}$
$\mathbf{y}$
$M.\texttt{drift\_detector()}$
No
Degradation Detected?
Yes
$\mathbf{x}$ collected from Δt before and after deployment
$M.\texttt{adapt()}$
Adapted Model $M_{adapted}$
Replace deployed model $M = M_{adapted}$
$\hat{\mathbf{y}}_{future}$
RUL Estimator
RUL
Assumed Δt interval
Assumed Δt interval
Time
Start collecting data
Model learned and deployed
Degradation started
Degradation detected
Model adaptation

Figure 1. The workflow describing how the adaptation technique could adapt the model trained in the controlled setting (nominal model) to predict RUL after the degradation in the system is detected.

driven, or hybrid (model-based + data-driven) approaches. We use standard techniques to train or build such a model M and train it to the input-output signals using the $M.\texttt{fit()}$ algorithm. Typically nominal $\mathbf{x}$ and $\mathbf{y}$ combinations can be used to build M. Once the model is trained, it is deployed, and the $M.\texttt{predict()}$ algorithm passes new $\mathbf{x}$ through M to generate new predicted $\mathbf{y}$. $M.\texttt{predict()}$, when fed with hypothesized future inputs $\hat{\mathbf{x}}_{future}$, can predict estimated future outputs $\hat{\mathbf{y}}_{future}$ which is then fed to the RUL Estimator which passes $\hat{\mathbf{y}}_{future}$ through a threshold function T to compute the RUL of the system.

Now, ideally, if there is no degradation in the system, the M model would forever be able to correctly predict the future observations of the system. However, this is never the case as all engineered systems eventually encounter some sort of degradation or failure. Also, there is no guarantee that the operating conditions will remain constant throughout the system's life. One way to adapt to this changing system dynamics would be to retrain the model for every new pair of $\mathbf{x}$ and $\mathbf{y}$; however, that process is computationally wasteful. Hence, to intelligently call the model update, we develop and deploy an $M.\texttt{drift\_detector()}$ algorithm that compares the predictions from M, and the sensors obtained from the real system to see if there is a statistically significant drift between the predicted and observed sensors. If yes, then while there are many reasons for which this drift could occur, we attribute this drift to degradations in the system that are not captured by the deployed model M anymore, and the parameters of this model need to be re-calibrated or adapted to the newly observed sensors. If that is the case, then we call the $M.\texttt{adapt}()$ that helps us adapt the model to the new dynamics using the newly observed data. We denote this adapted model as $M_{adapted}$. In our case, the JFR-based model adaptation algorithm is used to adapt the model to new data observed. $M_{adapted}$ now replaces the model M, and the process continues until a significant deviation is *again* detected in the sensor readings predicted by M and the observed sensor readings from the system. Figure 1 presents an overview of the overall workflow.

As established, the data from the controlled environments could be used to configure and train different models that represent the system. Over time, as the system's behavior changes, the model becomes stale and its predictions are inconsistent with the system's behavior. As the system operates in real life, different measurements are collected and stored in some database. Using the designed approach, there are two choices for model adaptation.

Condition-Based Model Adaptation (CBMA): The data are continuously collected from the deployment environment in this setting. The model and the system are constantly monitored, and any kind of deviations are detected and tagged. If the deviation is above the threshold, all the past information collected before the deviation happens is used for adapting the model. It is an offline adaptation technique as not all the incoming information is directly used for adaptation. Rather the adaptation is triggered based on the output of anomaly detection. From an implementation perspective, for condition-based model adaptation, we use data from a window comprising of Δt time steps *before and after* the time at which degradation was detected, where Δt is a design choice. The timeline at the bottom of Figure 1 visually depicts this.

Continuous Model Adaptation (CMA): In this scheme of adaptation, there is no dependency on the anomaly detection process. As and when a new measurement is recorded it is used for adapting the model. This helps in the continuous utilization of the incoming information.

While both CBMA and CMA methods enable the efficient use of the incoming data to update the model continuously, these two model adaptation methods have their pros and cons. Specifically, CMA requires high computing power as the models are continuously adapted. Furthermore, observing a single outlier can result in a deviation from the model's behavior, whereas it is not a persistent thing for the physical system. On the other hand, CBMA requires dependencies on the anomaly detectors and the storage systems where the data needed for model adaptation are stored. CBMA also enables us to quantify the model's behavior change which could be reflected based on the differences between the predictions of the nominal model that is trained using the data from a controlled setting, and the predictions of the adapted model that is adapted using incoming data predictions.

Our proposed adaptation approach can be used for either CMA or CBMA. Further, we have extended the offline model adaptation approach presented in Forgione et al. (2023) to hybrid models that represent the dynamics of a system by leveraging both first-principles domain knowledge and data-driven ML approaches. Karpatne, Watkins, Read, and Kumar (2017) presents physics-guided neural networks (PGNN), one such example of a hybrid modeling approach. PGNN uses the first principles model parallel to the data-driven components (e.g., RNNs). It could be helpful to directly use the first principle model even if it is not tuned and calibrated to the best quality. Such developed hybrid models for the specific systems could be further coupled with the adaptation technique to help us have a model that is always in close alignment with the physical system. A model that is always closely aligned with the physical system enables seamless deployment of different applications such as optimization, control, forecasting, prognostics and health management, automation, and decision-making, among others.

3. Experimental Setup and Results

Digital Synthetic Oilfield Testbed Setup: Our testbed's design is particularly chosen to mimic real-life oilfields. The test bed has three DC motor pumps attached to three flowmeters. Each DC motor pump pumps the water (used in place of oil) from the well to the eventual storage. The flowmeters measure the flow exiting the pumps. We also attached a fourth flowmeter to calculate the aggregated flow from the three pumps. Single and persistent faults are injected into each of the pumps to represent the loss of efficiency. The EOL condition for each pump is defined as the state when any pump's output flow dips below 0.15 units. The controllable input in the case of each pump is the pump speed, and the output measurement from the flowmeter would be the flow rate. Since the input voltage determines the pump speed, the voltage is considered the equivalent input variable for each of the pumps. Figure 2 represents the internal structure of the DC motor pump[1].

[1]https://ctms.engin.umich.edu/CTMS/?example=MotorSpeed\&section=SystemModeling

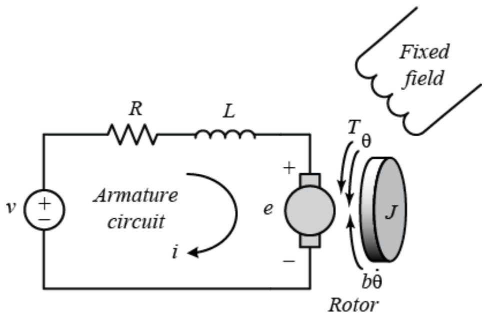


Figure 2. The electric equivalent circuit of a DC motor.

Based on the internal structure of each pump, the state-space model was designed for the testbed, considering the rotational speed and electric current as the state variables. After defining the established conditions to represent the system parameters, this state-space model for the testbed was used to simulate the operation of the three pumps in the oilfield. Equations (1a–1f) summarize the representation of this digital synthetic oilfield testbed. In this setup, (V_p for pump $p \in [1, 2, 3]$) represents the voltage and hence the controlled pump speed for each pump respectively (controllable inputs); ($y_p = \omega_p$, $p \in [1, 2, 3]$) represents the flow rate of each pump respectively (system measurements); and the hidden state variables include (ω_p, $p \in [1, 2, 3]$) that represents the angular momentum of each pump respectively, (i_p, $p \in [1, 2, 3]$) that represents the current drawn for each pump respectively. The inductance L_p, resistance R_p, and back electromotive force constant k_p for pump $p \in [1, 2, 3]$ are the system parameters.

$$\frac{d\omega_1}{dt} = \frac{1}{L_1}(V_1 - R_1 i_1 - k_1\omega_1) \tag{1a}$$
$$\frac{di_1}{dt} = \frac{1}{J_1}(k_1 i_1 - B_1\omega_1) \tag{1b}$$
$$\frac{d\omega_2}{dt} = \frac{1}{L_2}(V_2 - R_2 i_2 - k_2\omega_2) \tag{1c}$$
$$\frac{di_2}{dt} = \frac{1}{J_2}(k_2 i_2 - B_2\omega_2) \tag{1d}$$
$$\frac{d\omega_3}{dt} = \frac{1}{L_3}(V_3 - R_3 i_3 - k_3\omega_3) \tag{1e}$$
$$\frac{di_3}{dt} = \frac{1}{J_3}(k_3 i_3 - B_3\omega_3) \tag{1f}$$

There are multiple ways to model such systems. Neural state-space formulation is one such approach that could be used to model the system. This approach uses a couple of neural networks (NNs) to model state-transition and state-observation models. The same could be represented using Equations (2a–2b) where the function f represents the state-transition model and function g represents the state-observation model. In this neural state space formulation, once the model is trained, we

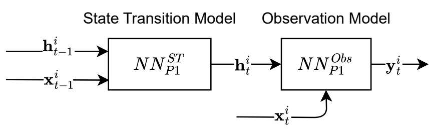


Figure 3. Schematic representation of neural state space model to model each component independently using just the inputs and hidden variable that affect it.

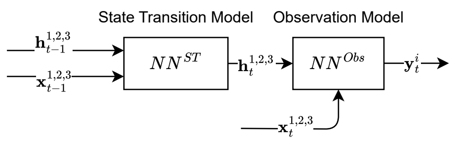


Figure 4. Schematic representation of neural state space model to model each component independently using all the inputs and hidden variables from the entire system.

utilize the forward Euler method to walk forward and make a closed-loop prediction for the next steps. Also note we are denoting the controllable inputs to the system by $\mathbf{x}$, the hidden state variables by $\mathbf{h}$, and the observed system measurements by $\mathbf{y}$. The subscript t represents the time step to which these values correspond.

$$\mathbf{h}_t = f(\mathbf{x}_{t-1}, \mathbf{h}_{t-1}) \tag{2a}$$
$$\mathbf{y}_t = g(\mathbf{x}_t, \mathbf{h}_t) \tag{2b}$$

Based on the described system for the testbed, we have a scenario where we have three pumps as the system's core components. For each of these three pumps, we have one controllable input and one measurement (output), and then internally, we have two state variables. Three major combinations were used for these different variables in our experiments. The summary and the description of why the particular selection was considered are described below.

1. One controllable input, one measurement, and three state variables: This setting enables us to represent each pump independently. Figure 3 represents this setting.
2. Three controllable inputs, one measurement, and seven state variables: In this setting, we model each pump's measurement independently but still consider all the input and underlying state variables. Figure 4 represents this setting. This enables us to consider all system dynamics together and learn the dependence of each pump's output on the entire system.
3. Three controllable inputs and three measurements: As shown in Figure 5, in this setting, we model the entire system using a single model that considers all the input variables and tries to learn the entire system by itself.

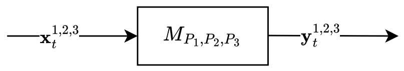


Figure 5. Schematic representation of the model when the entire system is modeled using the inputs and measurements.

In the first two settings, since we are able to use the available information for the state variables, the neural state space approach is used to model the system. There are existing approaches (e.g., Sheth, Roychoudhury, Chatar, and Celaya (2022)) that model the state-transition and state-observation function separately using two independent networks and in a joint setting where two networks are connected to each other. In this setting, these functions are represented using a NN. We also consider the measurement to be an unknown hidden state variable that needs to be estimated. In such a functional way, the state observation model becomes a passthrough function to select the state variable representing the measurement. Hence, the state-observation function could be omitted as represented in Equation 3. This particular method helps us condition the model in such a way that it has to produce the correct combination of the state variables as well as the measurement. By penalizing the wrong predictions, the model is regularized to adhere to the internal relations between state variables and the measurements. This could also be thought of as the state variables providing the regularization to the original model that is penalized for any sort of inconsistencies between the state variables and the measurement.

$$\mathbf{h}_t \cup \mathbf{y}_t = NN(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{y}_{t-1}) \tag{3}$$

The first two settings differ in the variables used by the NN to represent the state-space formulation. Equations 4 and 5 represent the two settings, respectively. The superscripted i represents the pump number to which different values correspond.

$$\mathbf{h}^i_t \cup \mathbf{y}^i_t = NN(\mathbf{x}^i_t, \mathbf{h}^i_{t-1}, \mathbf{y}^i_{t-1}); \quad i \in \{1,2,3\} \tag{4}$$

$$\mathbf{h}^i_t \cup \mathbf{y}^i_t = NN(\mathbf{x}^1_t, \mathbf{x}^2_t, \mathbf{x}^3_t, \mathbf{h}^1_{t-1}, \mathbf{h}^2_{t-1}, \mathbf{h}^3_{t-1}, \mathbf{y}^i_{t-1}); \quad i \in \{1,2,3\} \tag{5}$$

In the third setting, since the system is modeled as a whole, we utilize an LSTM network to model the system, and all the time dependence is captured through the LSTM network and can be represented as shown in Equation 6.

$$\mathbf{y}^1_t, \mathbf{y}^2_t, \mathbf{y}^3_t = LSTM(\mathbf{x}^1_t, \mathbf{x}^2_t, \mathbf{x}^3_t) \tag{6}$$

For all three settings, online and offline adaptation techniques have been integrated to adapt the model. Since the faults are injected independently into the pumps of the testbed, the summary table for the experimental results has four rows, one for the nominal setting and the remaining three for settings cor-

Table 1. Summary of experimental results (data-driven models). The metrics for all three pumps are shown separately under the columns for RMSE and R^2.

Scenario Number	Fault Setting	Number of Models	RMSE before Adaptation	RMSE after Adaptation	$\mathbf{R^2}$ before Adaptation	$\mathbf{R^2}$ after Adaptation
1	Nominal	3	[0.01, 0.01, 0.05]	-	[0.99, 0.99, 0.50]	-
	Fault in Pump 1	3	[0.06, 0.01, 0.05]	[0.01, 0.00, 0.01]	[-0.54, 0.99, 0.50]	[0.98, 0.99, 0.99]
	Fault in Pump 2	3	[0.01, 0.06, 0.05]	[0.00, 0.01, 0.01]	[0.99, -0.60, 0.50]	[0.99, 0.98, 0.99]
	Fault in Pump 3	3	[0.01, 0.01, 0.02]	[0.00, 0.00, 0.00]	[0.99, 0.99, 0.78]	[0.99, 0.99, 0.99]
2	Nominal	3	[0.01, 0.01, 0.02]	-	[0.98, 0.96, 0.92]	-
	Fault in Pump 1	3	[0.07, 0.01, 0.02]	[0.00, 0.00, 0.01]	[-0.98, 0.96, 0.92]	[0.99, 0.99, 0.98]
	Fault in Pump 2	3	[0.01, 0.07, 0.02]	[0.00, 0.01, 0.01]	[0.98, -1.22, 0.92]	[0.99, 0.99, 0.98]
	Fault in Pump 3	3	[0.01, 0.01, 0.05]	[0.00, 0.00, 0.01]	[0.99, 0.99, -0.04]	[0.99, 0.99, 0.98]
3	Nominal	1	[0.01, 0.02, 0.04]	-	[0.96, 0.95, 0.7]	-
	Fault in Pump 1	1	[0.06, 0.02, 0.04]	[0.01, 0.01, 0.01]	[-0.49, 0.95, 0.7]	[0.96, 0.99, 0.97]
	Fault in Pump 2	1	[0.01, 0.05, 0.01]	[0.01, 0.01, 0.01]	[0.96, -0.14, 0.7]	[0.99, 0.96, 0.97]
	Fault in Pump 3	1	[0.01, 0.02, 0.03]	[0.01, 0.01, 0.01]	[0.97, 0.95, 0.53]	[0.99, 0.99, 0.95]

Table 2. Summary of performance of physics-based model. The metrics for all three pumps are shown separately under the columns for RMSE and R^2.

System Setting	RMSE	$\mathbf{R^2}$
Nominal	[0.04, 0.04, 0.01]	[0.81, 0.83, 0.99]

responding to the faults in three pumps, respectively.

Table 1 summarizes the results from the experiments to model the system, where each model's performance is evaluated using the root mean square error (RMSE) and R^2 metrics, where

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{N}},$$

and

$$R^2 = 1 - \frac{\text{sum squared regression (SSR)}}{\text{total sum of squares (SST)}}.$$

For the sake of simplicity, the aggregated results from the online experiments are shown. Figure 6 shows the plots depicting the nominal model's prediction, predictions from the adapted model, and the actual system behavior when the fault was present in Pump 1. It also shows the threshold value, which could be used to predict RUL. Essentially, the time when the flow value for any pump goes below the threshold could be considered as the RUL for the system.

(Karpatne et al., 2017) introduced PGNN that enables us to couple the physics-based model with an NN. The core idea behind the coupling is to allow the NN to overcome the regions where the physics-based model might make errors because of the lack of generalizability introduced by the poorly estimated parameters. (Sheth et al., 2022) have successfully demonstrated the advantages of integrating the PGNN with neural state-space models. Figure 7 represents the original structure of the PGNN and Figure 8 represents the modified structure of PGNN for the neural state-space model as designed in (Sheth et al., 2022)

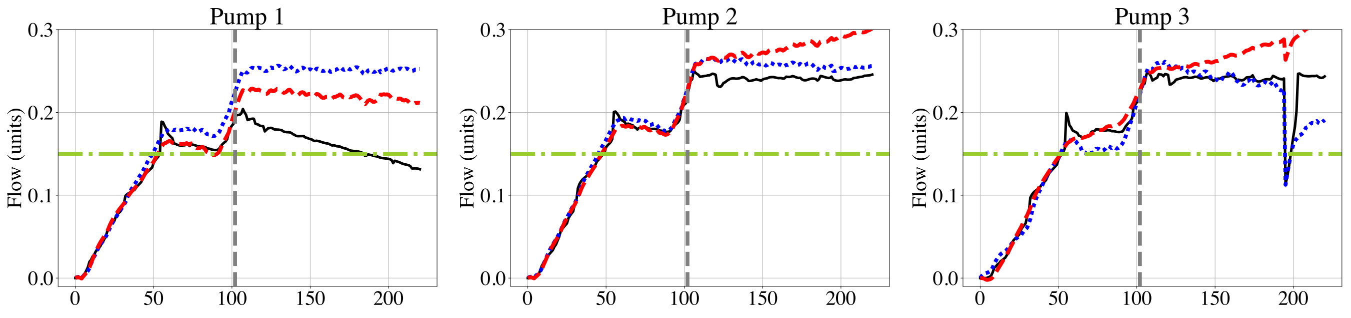
Inspired by these works and the ensuring need for the generalizability and scientific accuracy of the models representing the system, we have implemented all three scenarios using hybrid models that combine physics-based models with data-driven models. Since we designed the testbed, we can access the actual physics model used to collect the data. However, to mimic the scenarios we have in real life where the exact physics model is unavailable, we decided to use the functional form of the original physics model. We estimated the parameters after introducing some random noise to the recorded measurements. In doing so, we estimated the parameters of the physics model, which were not completely aligned with the underlying system model. This mimics the scenario of having a physics-based model that is not well-calibrated. Table 2 summarizes the performance of the physics-based model in the nominal setting.

Based on the description of our neural state-space model and the PGNN architecture, there are two major ways in which the output from the physics model could be used:

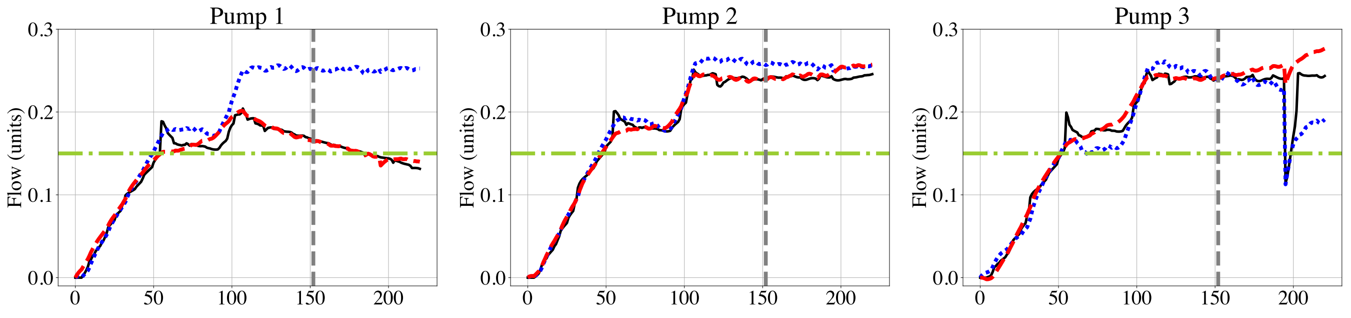
1. Using the output estimate from the physics model as an input to the data-driven model. This strictly represents the PGNN architecture described in Figure 7. Figure 9 represents the same in our scenario.
2. Treating the output estimate from the physics model as one of the state-transition variables in the neural state-space formulation. This way, the output estimate from the physics model is integrated into the state-transition part of the neural state-space model. The output estimate from the physics model from the previous timestep is considered while predicting the actual output for the current timestep. Also, the model is penalized for pre-

Table 3. Summary of experimental results (physics-regularized data-driven models). The metrics for all three pumps are shown separately under the columns for RMSE and R^2.

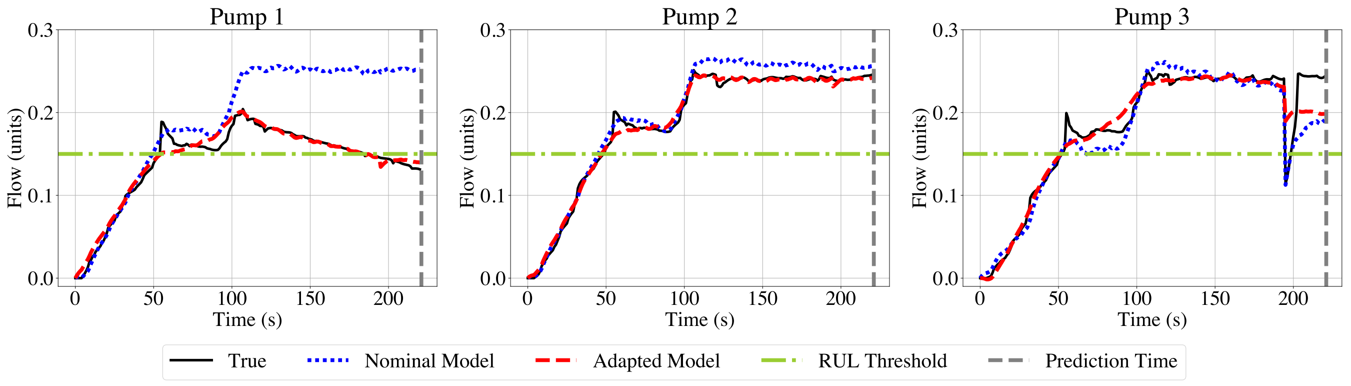
System Setting	**RMSE**	**RMSE after Adaptation**	**R^2**	**R^2 after adaptation**
Nominal	[0.01, 0.02, 0.02]	-	[0.99, 0.94, 0.90]	-
Fault in Pump 1	[0.06, 0.02, 0.02]	[0.01, 0.01, 0.01]	[-0.71, 0.94, 0.90]	[0.98, 0.99, 0.99]
Fault in Pump 2	[0.01, 0.08, 0.02]	[0.01, 0.01, 0.01]	[0.99, -1.46, 0.90]	[0.99, 0.99, 0.99]
Fault in Pump 3	[0.01, 0.02, 0.06]	[0.01, 0.01, 0.02]	[0.99, 0.94, 0.38]	[0.99, 0.99, 0.90]



(a) Prediction done at time 102. The data before time 102 is used to adapt the models.



(b) Prediction done at time 152. The data before time 152 is used to adapt the models.



(c) Prediction done at time 221 (End of the simulation). The data before time 221 is used to adapt the models.

Figure 6. Flow estimates for the three pumps of the system representing the oilfield using the data from different timesteps to adapt the model. The blue dotted line represents the estimates from the nominal model, the red dashed line represents the estimates from the model after adaptation, and the black solid line represents the actual system behavior when the fault has been introduced in Pump 1. The green dashdot horizontal line represents the threshold that could be used to determine the RUL of the system. The vertical gray dashed line represents the present time till which the data from the system are observed.

dicting the wrong value for the output estimate from the physics model. Thus, the output estimate for the physics model works both as a signal for predicting the system's output from the model and provides a regularization effect for the model to be grounded to the physical relationships captured by the physics model. We refer to this set-

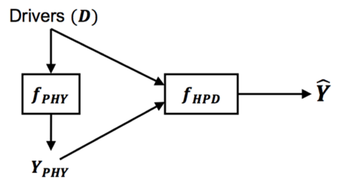


Figure 7. Structure of PGNN (Karpatne et al., 2017).

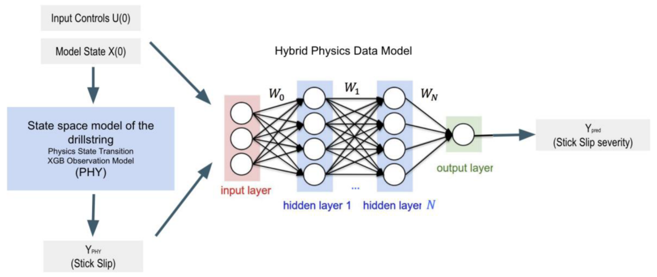


Figure 8. Schematic representation of PGNN for neural state-space model as shown in (Sheth et al., 2022).

ting as physics-regularized neural network (PRNN) and is shown in Figure 10. The predicted physics model estimate is compared with the actual physics model estimate. It is added to the loss function for training, and in the inference phase, we can ignore this output.

In our experiments, we evaluated both techniques and we coupled the physics model with the neural state-space models that we have for the second scenario. Table 3 summarizes the results from the experiments for the PRNN model corresponding to the second setup where the output of the physics model is used as a regularization condition. Figure 11 showcases the prediction estimates when the nominal version of the PRNN model is used for forward Euler simulation to generate the predictions for all timesteps in the closed loop setting and the prediction estimate for the same system state with the adapted PRNN model. Conducting experiments with PRNNs in this particular setting validates how the adaptation technique has been integrated into the neural state-space model, thus enabling the adaptation of hybrid models representing the system. Similar to the neural state-space model, the estimates of the physics-based model could be integrated into the list of controllable inputs for the LSTM model. When experiments for this particular setting were conducted, similar to the neural state-space model, positive results were obtained.

Based on the presented results in Table 1 and Figure 6, it is evident that the adaptation technique helps robustly adapt to the fault scenario. Further, it helps slightly reduce the errors due to the noise in measurements for the components without any faults, thus improving the overall prediction. Comparing the performance of the nominal model learned using the physics with the PRNN (first row of Table 2 and Table 3), it could be seen that the PRNN helps improve the performance over

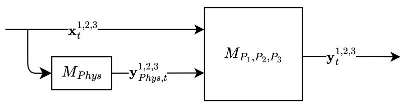


Figure 9. Schematic representation of PGNN for neural state-space model for our testbed.

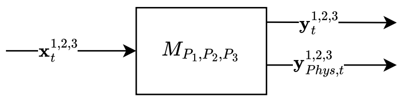


Figure 10. Schematic representation of PRNN for neural state-space model for our testbed.

the model learned just using the physics by eliminating the error resulting from the suboptimally estimated parameters from the physics-based model. Comparing the performance of the data-driven model and PRNN from Table 1 and Table 3, it could be observed that the PRNN model helps ensure the model follows the underlying physics and, hence, could help improve the performance of the data-driven model. The difference in the performance of nominal models for Pumps 1 and 2 between the data-driven model and PRNN is not significant due to the simplicity of the components. However, the improvement is evident in the case of Pump 3, which was set slightly differently to drop its performance instantaneously after 3 minutes and then behave normally again. In this case, the data-driven model's performance degrades as this noisy instance hurts the model's training. However, since the PRNN obtained the signal from the physics model, it was able to stay on track with the training process. Further, based on Table 3, and Figure 11, it is clear that the adaptation technique successfully adapts the learned PRNN model to faulty scenarios.

To estimate the amount of Carbon Dioxide (CO_2) produced by the cloud or personal computing resources used to execute the code, one may use the `Code Carbon`[2] library. It helps us track the carbon emissions for any computational process by considering the region where the machine is located, the amount of CPU and GPU consumed, the power used to run the particular process, and the overall machine's power consumption. By tracking all of this, the library can compute an estimate of the carbon intensity and energy consumption, thus resulting in the final number representing the CO_2 emissions.

Based on this library, Table 4 summarizes the power used as well as the CO_2 emissions in the process to train the model, retrain it using the standard transfer learning approach, and the adaptation of the model using the JFR method. For conducting this study, experiments were hosted to the Google Cloud Platform so as to have an accurate track of the compute resources as well as the carbon intensity. Using the local computer, the variables such as the particular power source

[2]`https://codecarbon.io/`

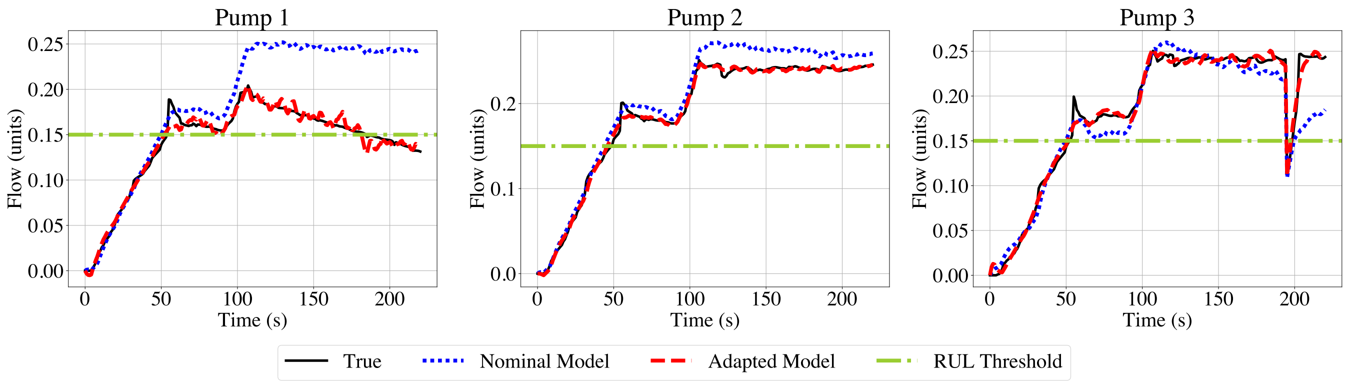


Figure 11. Flow estimates for the three pumps of the system representing the oilfield. The estimates are derived after running the forward Euler simulation on the PRNN model. The blue dotted line depicts the estimates derived using the model before adaptation (i.e.) in the nominal state. The red dashed line represents the estimates from the model after adaptation, and the black solid line represents the actual system behavior when the fault has been introduced in Pump 1. The green dashdot horizontal line represents the threshold that could be used to determine the RUL of the system.

being used and other local factors can skew the results.

Table 4. Summary of carbon emissions and power usage.

Phase	Carbon Emissions	Energy Consumed
Training	1.70×10^{-4}	2.84×10^{-3}
Retraining	1.70×10^{-4}	2.8×10^{-3}
Adaptation	1.37×10^{-6}	2.28×10^{-5}

From the results in Table 4, it is evident that the process of adaptation results in far lower levels of carbon emissions as well as less power usage. This is a clear advantage of using the adaptation technique instead of the standard retraining-based transfer learning as the carbon emissions are reduced, improving the sustainability aspects of the developed solution; the duration for which both processes are run is also much different where adaptation is approximately 10 times faster and uses far less computational resources.

4. Conclusions and Future Work

In this paper, we presented an automated online model adaptation framework for robust RUL prediction. We showcased the JFR-based adaptation technique to adapt the models representing the system for online RUL prediction and also extended this technique to adapt the hybrid ML approaches that provide robust system representation. The results indicate that this approach is much more computationally efficient than retraining a data-driven model based on standard transfer learning methods. In the future, we would like to continue modifying the algorithms to relax some of the assumptions. We would also like to extend this approach to switched hybrid systems that combine discrete modes along with continuous system dynamics in each discrete mode.

References

Cheng, H., Kong, X., Wang, Q., Ma, H., Yang, S., & Chen, G. (2023). Deep Transfer Learning Based on Dynamic Domain Adaptation for Remaining Useful Life Prediction Under Different Working Conditions. *Journal of Intelligent Manufacturing*, *34*(2), 587-613.

da Costa, P. R. d. O., Akçay, A., Zhang, Y., & Kaymak, U. (2020). Remaining Useful Lifetime Prediction via Deep Domain Adaptation. *Reliability Engineering & System Safety*, *195*, 106682.

Ding, Y., Ding, P., Zhao, X., Cao, Y., & Jia, M. (2022). Transfer Learning for Remaining Useful Life Prediction Across Operating Conditions Based on Multi-source Domain Adaptation. *IEEE/ASME Transactions on Mechatronics*, *27*(5), 4143-4152.

Forgione, M., Muni, A., Piga, D., & Gallieri, M. (2023). On the Adaptation of Recurrent Neural Networks for System Identification. *Automatica*, *155*, 111092.

Huang, Z., Xu, Z., Wang, W., & Sun, Y. (2015). Remaining Useful Life Prediction for a Nonlinear Heterogeneous Wiener Process Model With an Adaptive Drift. *IEEE Transactions on Reliability*, *64*(2), 687-700.

Karpatne, A., Watkins, W., Read, J., & Kumar, V. (2017). Physics-Guided Neural Networks (PGNN): An Application in Lake Temperature Modeling. *arXiv preprint arXiv:1710.11431*, *2*.

Lei, Y., Li, N., Gontarz, S., Lin, J., Radkowski, S., & Dybala, J. (2016). A Model-Based Method for Remaining Useful Life Prediction of Machinery. *IEEE Transactions on Reliability*, *65*(3), 1314-1326.

Liu, J., Saxena, A., Goebel, K., Saha, B., & Wang, W. (2010). An Adaptive Recurrent Neural Network for Remaining Useful Life Prediction of Lithium-ion Batteries. In *Annual conference of the PHM Society* (Vol. 2).

Liu, L., Guo, Q., Liu, D., & Peng, Y. (2019). Data-Driven

Remaining Useful Life Prediction Considering Sensor Anomaly Detection and Data Recovery. *IEEE Access*, *7*, 58336-58345.

Ma, M., & Mao, Z. (2021). Deep-Convolution-Based LSTM Network for Remaining Useful Life Prediction. *IEEE Transactions on Industrial Informatics*, *17*(3), 1658-1667.

Pan, D., Li, H., & Wang, S. (2022). Transfer Learning-Based Hybrid Remaining Useful Life Prediction for Lithium-Ion Batteries Under Different Stresses. *IEEE Transactions on Instrumentation and Measurement*, *71*, 1-10.

Sheth, P., Roychoudhury, I., Chatar, C., & Celaya, J. (2022). A Hybrid Physics-Based and Machine-Learning Approach for Stick/Slip Prediction. In *SPE/IADC Drilling Conference and Exhibition.*

Si, X.-S., Hu, C.-H., Chen, M.-Y., & Wang, W. (2011). An Adaptive and Nonlinear Drift-based Wiener Process for Remaining Useful Life Estimation. In *2011 Prognostics and System Health Management Conference* (p. 1-5).

Siahpour, S., Li, X., & Lee, J. (2022). A Novel Transfer Learning Approach in Remaining Useful Life Prediction for Incomplete Dataset. *IEEE Transactions on Instrumentation and Measurement*, *71*, 1-11.

Sun, C., Ma, M., Zhao, Z., Tian, S., Yan, R., & Chen, X. (2019). Deep Transfer Learning Based on Sparse Autoencoder for Remaining Useful Life Prediction of Tool in Manufacturing. *IEEE Transactions on Industrial Informatics*, *15*(4), 2416-2425.

Wang, Y., Zhao, Y., & Addepalli, S. (2020). Remaining Useful Life Prediction using Deep Learning Approaches: A Review. *Procedia Manufacturing*, *49*, 81-88.

Zhang, A., Wang, H., Li, S., Cui, Y., Liu, Z., Yang, G., & Hu, J. (2018). Transfer Learning with Deep Recurrent Neural Networks for Remaining Useful Life Estimation. *Applied Sciences*, *8*(12).

Zhang, Y., Xiong, R., He, H., & Liu, Z. (2017). A LSTM-RNN Method for the Lithuim-ion Battery Remaining Useful Life Prediction. In *2017 Prognostics and System Health Management Conference (PHM-Harbin)* (p. 1-4).

Zhang, Y., Yang, Y., Xiu, X., Li, H., & Liu, R. (2021). A Remaining Useful Life Prediction Method in the Early Stage of Stochastic Degradation Process. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *68*(6), 2027-2031.

BIOGRAPHIES

Prasham Sheth is a Data Scientist at SLB Software Technology Innovation Center. His research interests include the application of machine learning, deep learning, and hybrid modeling-based approaches to solving complex problems in computer vision and time-series analysis. He received a Master of Science in Data Science from Columbia University, New York, New York, USA, and a Bachelor of Technology in Computer Engineering from Nirma University, Ahmedabad, Gujarat, India.

Indranil Roychoudhury is a Principal AI Scientist at SLB Software Technology Innovation Center, and his primary area of research is time-series analysis by combining physics-based approaches with machine learning approaches. He holds his Ph.D. and M.S. in Computer Science from Vanderbilt University and was a Senior Research Scientist at NASA Ames Research Center before joining SLB. He is a Fellow of the Prognostics and Health Management Society and a Senior Member of IEEE.