A Generative Reinforcement Learning Framework for Predictive Analytics

Erotokritos Skordilis, Ph. D., University of Miami
Ramin Moghaddass, Ph. D., University of Miami
Md Tanzin Farhat, University of Miami

Key Words: predictive analytics, variational autoencoders, reinforcement learning, remaining useful life

SUMMARY

In this work, we present a new approach for latent system dynamics and remaining useful life (RUL) estimation of complex degrading systems using generative modeling and reinforcement learning. The main contributions of the proposed method are two-fold. First, we show how a deep generative model can approximate the functionality of high-fidelity simulators and, thus, is able to substitute expensive and complex physics-based models with data-driven surrogate ones. In other words, we can use the generative model in lieu of the actual system as a surrogate model of the system. Furthermore, we show how to use such surrogate models for predictive analytics. Our method follows two main steps. First, we use a deep variational autoencoder (VAE) to learn the distribution over the latent state-space that characterizes the dynamics of the system under monitoring. After model training, the probabilistic VAE decoder becomes the surrogate system model. Then, we develop a scalable reinforcement learning framework using the decoder as the environment, to train an agent for identifying adequate approximate values of the latent dynamics, as well as the RUL.

To our knowledge, the method presented in this paper is the first in industrial prognostics that utilizes generative models and reinforcement learning in that capacity. While the process requires extensive data preprocessing and environment tailored design, which is not always possible, it demonstrates the ability of generative models working in conjunction with reinforcement learning to provide proper value estimations for system dynamics and their RUL. To validate the quality of the proposed method, we conducted numerical experiments using the *train_FD002* dataset provided by the NASA CMAPSS data repository. Different subsets were used to train the VAE and the RL agent, and a leftover set was then used for model validation. The results shown prove the merit of our method and will further assist us in developing a data-driven RL environment that incorporates more complex latent dynamic layers, such as normal/faulty operating conditions and hazard processes.

1 INTRODUCTION

The increased complexity and inherent uncertainty of modern sensor-intensive systems present a great challenge for accurately capturing their hidden dynamics and degradation that are necessary for control and decision-making applications such as *remaining useful life* (RUL) estimation. A typical approach requires the development of high-fidelity physics-based model simulators that, due to their design, can efficiently simulate system dynamics. However, such systems can be very expensive to create (e.g., digital twins), and their application is constrained only to a small number of industrial systems where the dynamics are understood to a high detail. Because of the lack of proper simulators for most systems in contemporary industry, researchers prefer to develop data-driven approaches, such as state-space models (SSM). SSMs, among other things, are attractive approximators due to their ability to quantify the inherent uncertainty of system dynamics. Estimating latent dynamics is usually conducted via Bayesian inference, where tools such as Kalman and particle filters and Bayesian statistics tools such as Markov Chain Monte Carlo methods are most prominent. These approaches, nevertheless, incorporate a significant level of inductive bias caused by significant limitations due to unrealistic parametric and distributional assumptions, high computational complexity of latent state estimation, and the requirement of very large volumes of training data that are not always easy to obtain. For these reasons, instead of state-space models, our approach works by designing a system simulator that uses deep generative modeling, and more specifically variational autoencoders (VAE). Because of their characteristics, such models can approximate any system's dynamics with arbitrary accuracy, making them a good candidate for a surrogate simulator when the actual one is not accessible. Combining generative models with state-of-the-art reinforcement learning (RL) techniques, surrogate simulators show great promise in addressing many of the issues pertaining to state-space described above for modeling a system's dynamics, since they do not require any predefined mathematical model or distributional assumption.

The method proposed in this paper focuses on estimating RUL values for a set of simulated turbofan engines from the NASA C-MAPSS dataset [1] using a non-parametric function approximator instead of a predefined state-space model, that 1) is designed to work as a surrogate simulator of the actual system, and 2) generates controlled observations through agent

2023 Annual Reliability and Maintainability Symposium (RAMS) | 978-1-6654-6053-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/RAMS51473.2023.10088246

actions sampled from a stochastic policy. The VAE decoder model is used as a dynamic environment with which the RL agent interacts through proper actions to produce RUL estimates. Although our method is not a pure model-based RL, we show that when the environment is represented by a generative surrogate model, it can be very sample efficient thus requiring a lot less training data.

2 LITERATURE REVIEW

Despite VAEs and RL being at the forefront of applied research in many different areas, literature examples that combine them are still scarce and utilized only in a handful of applications. Particularly for industrial prognostics and health management (PHM), there are no such published works. A recently published work for combinatorial optimization can be found in [2], where the author used VAEs for path identification on transportation graphs and then used RL for end-to-end optimization of the proposed framework. In [3], the authors investigated the usage of generative models and RL for safe machine learning. More specifically, they used VAEs and generative adversarial networks to eliminate threats of adversarial attacks on RL agents. In another example presented in [4], which shares some similarities to our work but for a different domain, the authors proposed a transitional VAE as an environment model for an RL agent, where the objective was to learn an optimal patient treatment policy. Except for VAEs, deterministic autoencoders have also been used in improving sample efficiency in off-policy RL methods [5]. The authors used a deterministic autoencoder to augment a soft actor-critic RL method, leading to a stable agent training using state representations of lower dimensionality and, thus, improved computational times. Another approach that combines VAEs and RL is presented in [6]. In this case, a VAE was used to learn disentangled latent representations of action sequences provided by experts, and then the trained decoder was used as an augmented agent in an RL setting to perform a search more efficiently in the action space. Another example of using VAEs for safe RL can be found in [7], where the authors combined VAEs, risk-directed exploration, and curiosity to train deep Q-networks using imaginary future state trajectories for autonomous vehicles.

While these works show the advantages of combining generative models with RL when the state and action spaces are very large, there are currently no similar works in prognostics and diagnostics for complex industrial systems. Our intention with the proposed method is, therefore, to present the first such example in the predictive analytics literature.

3 DEEP GENERATIVE MODELING

The main objective in predictive analytics applications is the inference of latent system dynamics using time-varying observations collected from arrays of sensors. Most applications call for the development of multi-stage data-driven SSMs that consist of mathematical formulas that map the observations to latent states (e.g., degradation), as well as describe the Markovian relationship of successive states, i.e., $\Pr(\boldsymbol{y_t}|\boldsymbol{x_t})$, $\Pr(\boldsymbol{x_t}|\boldsymbol{x_{t-1}})$, where $\Pr(\cdot\,|\,\cdot)$ denote conditional probability densities, and $\boldsymbol{y_t}, \boldsymbol{x_t}$ denote sensor observations and latent dynamics, respectively. SSMs are very effective in describing system dynamics, but there are cases where they fail to capture them, especially when the system dynamics are governed by high-dimensional non-linear models. Therefore, non-parametric function approximator methods seem to be reasonable alternative approaches, particularly because they do not require any predefined structure for the conditional densities. Neural networks that are considered "universal function approximators" can be used as non-parametric densities instead of parametric formulations, and they can characterize the system dynamics. In this paper, we aim to model system dynamics using fully non-parametric function approximators in the form of variational autoencoders (VAEs).

In its simplest setting, a VAE defines a generative model as the following joint distribution:

$$\Pr_\theta(\boldsymbol{z}, \boldsymbol{y}) = \Pr_\theta(\boldsymbol{z}) \Pr_\theta(\boldsymbol{y}|\boldsymbol{z}), \tag{1}$$

where θ is the set of the neuron weights. Apart from that, VAEs also fit a recognition model (or inference network) as:

$$q_\phi(\boldsymbol{z}|\boldsymbol{y}) \approx \Pr_\theta(\boldsymbol{z}|\boldsymbol{y}). \tag{2}$$

Therefore, the overall model can be analyzed into an encoder model that encodes the inputs $\boldsymbol{y}$ (i.e., sensor observations) into a stochastic latent bottleneck $\boldsymbol{z}$, and a decoder model that reconstructs the input (Figure 1).

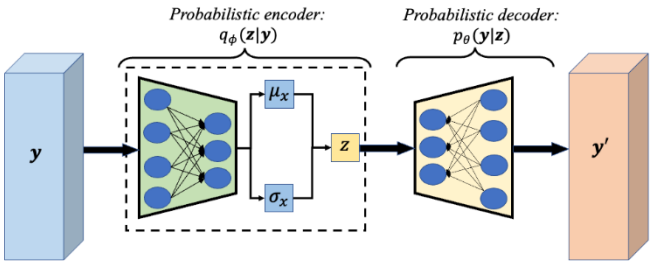

Figure 1: Variational autoencoder. The probabilistic encoder encodes the input to a latent layer, and the decoder samples from that layer to reconstruct the input

Fitting the model requires maximizing the marginal likelihood:

$$\Pr_\theta(\boldsymbol{y}) = \int \Pr_\theta(\boldsymbol{y}|\boldsymbol{z}) \Pr_\theta(\boldsymbol{z}) d\boldsymbol{z}. \tag{3}$$

Because computing the marginal likelihood is intractable, we use the inference network to compute an approximate posterior $q_\phi(\boldsymbol{z}|\boldsymbol{y})$, where ϕ denotes the set of inference network's neuron weights. This can be utilized to compute the *evidence lower bound (ELBO)*:

$$\mathcal{E}(\theta, \phi\,|x) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{y},\phi)}[\log \Pr_\theta(\boldsymbol{y}|\boldsymbol{z})] - \mathbb{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{y})||\Pr(\boldsymbol{z})). \tag{4}$$

ELBO can be viewed as the summation of the expected log-likelihood and a regularization term that penalizes large differences between the posterior and the prior. Using stochastic gradient descent, ELBO can be maximized with respect to θ, ϕ and train the two networks. However, while it is easy to take gradients for θ, the same cannot be said for ϕ, since

the sampling process in the latent layer depends on it. For that reason, VAE training utilizes the *reparameterization trick*, where we modify $\boldsymbol{z}$ as $\boldsymbol{z} = \mu_x + \sigma_x \odot \epsilon$, with $\epsilon \sim N(0, \boldsymbol{I})$. The ELBO then becomes:

$$\mathcal{E}(\theta, \phi \mid x) = \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{y},\phi)}[\log \Pr_{\theta}(\boldsymbol{y}|\boldsymbol{z})]\mathbb{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{y}) || \Pr(\boldsymbol{z})). \quad (5)$$

Now, the expectation is independent of the weight parameters and gradients can flow backward using backpropagation, leading to network training.

4 REINFORCEMENT LEARNING

From a high-level perspective, the goal of reinforcement learning (RL) is to learn a policy for generating a sequence of actions that optimizes the expected discounted future returns when starting from any given state. A distinguishing characteristic of RL is that the information needed to identify such a sequence can be acquired via trial-and-error interaction with the target system (i.e., *model-free RL*). RL algorithms aim to approximate a policy function π that directly maps environment states into actions, $a_t \sim \pi(s_t)$. We can, therefore, define an optimal policy π^* from which an RL agent samples actions that maximize the expected return, or cumulative future reward, when beginning from a state s_t and following π^* as:

$$\pi^* = \arg max_\pi \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)], \quad (6)$$

where $\gamma \in [0,1]$ is a discount factor, $r(s_t, a_t)$ is the reward received from traversing to state s_t using action a_t, and $\mathbb{E}_\pi[\cdot]$ is the expected return by following policy π up towards the terminal state. Due to the combinatorial nature of the state space of real-world systems, modern RL algorithms follow parameterized policies where function approximators are used. The most popular choice of such a policy is a neural network parameterized by neuron weights ξ. We denote this dependency by $\pi(\xi)$. For parameterized policies, the above optimization problem can then be recast as:

$$\xi^* = \arg max_\xi \mathbb{E}_{\pi_\xi}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]. \quad (7)$$

Most RL algorithms directly model the value function ($\mathcal{V}_\pi(s_t) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_t]$), or the action-value function ($\mathcal{Q}_\pi(s_t, a_t) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_t, a_t]$), which calculates the expected return. This expression works as a basis for many RL algorithms since the agent's objective is to learn to take actions that maximize $\mathcal{Q}$, and as a result, learn an optimal policy. Similar to the policy, both the value and action-value functions can also be parameterized.

Generally, RL algorithms can be categorized based on which functions they approximate as: a) value-function based, b) policy gradient based, and c) actor-critic based. The latter is a hybrid method that uses both a policy network that provides actions (actor) and a value network (critic) that returns the expected return from following the proposed actions. In this paper, we used an actor-critic method, namely Proximal Policy Optimization [8].

4.1 *Model-based RL vs. Model-free RL vs. Proposed method*

Here, we need to emphasize another novelty and contribution of the proposed method, compared to other RL-based applications. With respect to the environment, the agent interacts with, there are two types of RL methods, namely *model-free* and *model-based*. The former does not require any knowledge of the environment dynamics, which are purely probabilistic, but rather the agent learns through experience collected from a direct exposure of the agent to its environment through trial-and-error. While dynamics models are easy to define for environments with a small number of states and actions, defining and storing a dynamics model for real-world applications is challenging due to the sheer size of their state and action spaces. Model-free RL methods do not require such models for agent training, so they are easier to implement for real-world applications. However, without a dynamics model, the agent will need to try a very large number of state/action combinations until it learns a good policy. Therefore, model-free methods are sample inefficient, and they need to go over many trial-and-error iterations, leading to long training times and slow convergence. On the other hand, model-based RL methods are distribution models, because they produce a description of all possibilities and their respective probabilities $Pr\ (s_t|s_{t-1}, a_t)$. Since in real-world applications this is not easy, model-based RL methods rely on producing simulated experience using planning, a computational process that uses the (proposed) model as input and produces/improves a policy via interactions with the simulated environment. Because of that, model-based RL methods are very sample efficient since they can produce their own "fake" data to train the agent, but they are also more complex compared to model-free approaches.

Our proposed method stands in the middle of these two types of RL. We consider the pretrained decoder described earlier as a probabilistic model for the environment, but we do not perform any form of planning at this stage. The results obtained show us that we can obtain satisfactory RUL estimates for previously unseen degrading systems that follow the same distributional assumptions with the systems used to train the agent, by utilizing a model-free RL method for agent training (Section 4.2), and at the same time avoiding simulated rollouts of the environment (planning).

4.2 *Proximal Policy Optimization*

Proximal Policy Optimization (PPO) is an actor-critic based RL method that has proven to be highly effective for optimizing large non-linear policies. Apart from that, we chose PPO due to its monotonic improvement properties guaranteed by Kullback-Leibler (KL) divergence on policy updates. PPO aims to optimize the following unconstrained optimization problem:

$$\max_\xi \mathbb{E}_t \left[\frac{\pi_\xi(a_t|s_t)}{\pi_{\xi_{old}}(a_t|s_t)} \hat{A} - \beta KL\left[\pi_{\xi_{old}}(\cdot\,|s_t), \pi_\xi(\cdot\,|s_t)\right] \right], \quad (8)$$

with $\beta, \hat{A}$ being a penalty coefficient and an estimator of the advantage function, respectively. PPO computes an update at each step that minimizes the cost function, while ensuring that the deviation from the previous policy remains relatively small. To avoid large policy updates that could lead to training instabilities, PPO clips the ratio between the current and old

policies using a manually tuned hyperparameter ϵ as:

$$L_{\xi} = \mathbb{E}_t\left[\frac{\pi_{\xi}(a_t|s_t)}{\pi_{\xi_{old}}(a_t|s_t)}\hat{A}\right] = \mathbb{E}_t\left[r_t^{\xi}\hat{A}\right] \Longrightarrow,$$

$$L_{\xi}^{CLIP} = \mathbb{E}_t[min\,(r_t^{\xi}\hat{A}, clip\left(r_t^{\xi}, 1-\epsilon, 1+\epsilon\right)\hat{A}\Big]. \quad (9)$$

For more details regarding PPO the interested reader should refer to the original manuscript [8].

5 PROPOSED METHODOLOGY

The purpose of the method described in this paper is to present a novel predictive analytics approach focusing on RUL estimation of degrading systems governed by high-dimensional complex system dynamics. To achieve this, we use a two-pronged method that utilizes VAEs and RL in the manner described in this Section.

5.1 *Surrogate system model*

In the first step of our approach, we utilize data of failed systems to train the VAE. These data can be categorized as sensor readings, control inputs, and true lifetimes. At the end of the training process, the probabilistic decoder is ready to be used as a surrogate simulator for the actual system under observation. At this point the decoder, sampling from the encoded layer that follows a predefined distribution, can generate new data without the need to access the true simulator, at a fraction of the cost. This is a side-product of our work that can be utilized to augment small datasets, without the need to collect true observations, a process that is expensive and challenging. However, the most important outcome at this point is that the probabilistic decoder can be used as an environment with which an RL agent can interact to learn a policy that leads to RUL value prediction. In the experiments conducted we considered a two-dimensional latent state, however any dimension size can be used.

5.2 *RL-based RUL estimation*

After VAE training, we proceed with the second step of the proposed method, where we train the RL agent. During RL training, the agent samples actions from a stochastic policy, which are provided as inputs to the environment-decoder. The system then traverses to a new state and generates a numerical reward that reflects the quality of the action taken. The new state and the reward are given as feedback to the agent, which asserts the quality of the action taken. By optimizing the returned cumulative reward, the agent manages to learn the optimal policy.

For the purposes of the proposed method, the agent learns a policy whose sampled actions follow the same distributional assumptions as the VAE's encoded layer. In other words, the actions represent latent representations of the system dynamics observed during its lifetime and the agent's objective is to learn to control samples from the encoded layer. Mathematically, a new system state $\boldsymbol{y}_{t+1}$ is generated as:

$$a_t \sim \pi(\boldsymbol{y}_t), \quad (10)$$

$$\boldsymbol{y}_{t+1} \sim Pr\,(\boldsymbol{y}_t|a_t). \quad (11)$$

The agent training proceeds as follows: At every timestep, an action is sampled from the policy and is given as input to the environment-decoder. A forward pass then occurs and the environment outputs the RUL estimated values. The quality of those values is measured through comparing them with the true values using mean squared error (MSE), which in our case functions as the returned reward. The main RL training objective is, thus, the minimization of the MSE error between true observations and reconstructed decoder outputs. Other standard loss measures such as root mean squared error (RMSE) or mean absolute error (MAE) can also be used, depending on the task at hand, but they are not used here.

The main purpose of the trained RL agent is to generate the necessary actions that, when fed to the decoder, the latter can estimate RUL values for systems that are still operational (i.e., testing set). Figure 2 presents a flowchart of the proposed RUL estimation method.

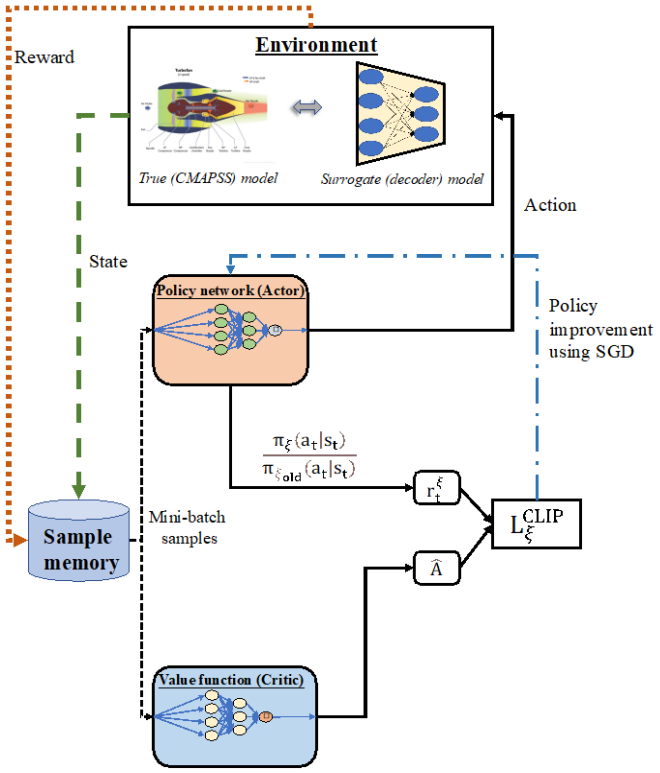

Figure 2: RL-based RUL estimation

6. NUMERICAL EXPERIMENTS

To assess the validity of the proposed methods, we conducted experiments using the *train_FD002* NASA C-MAPSS dataset. The dataset contains simulated trajectories of run-to-failure sensor observations and operating conditions for a fleet of 260 turbofan engines in a time-series format. Furthermore, all engines begin their function from an unknown level of wear & tear. To accommodate for the training and evaluation of all models in the proposed structure, the dataset was divided into three mutually exclusive subsets:

- Units 1-200: Training data for the VAE.
- Units 201-240: Training data for the RL agent.
- Units 241-260: Test set for the evaluation.

Because the sensors and operating inputs are given in various value scales, we performed a normalization of all values

between zero and one. Another data preprocessing step followed was to further smooth the input data using cumulative moving averages to consider time dependencies between successive observations. Nevertheless, this is an assumption that normally should be avoided, and thus it is an assumption that we want to address in future work. Also, apart from the time-series data provided by the CMAPSS dataset, we also included a column for the lifetimes of all engines given as a percentage of their RUL in the data, e.g., for an engine with lifetime equal to *10* operation cycles, the RUL at time $t = 1$ is equal to 90% of operational life left. The reason behind using RUL estimates instead of actual lifetimes was that the values given in that format are also between zero and one, and therefore, they are in a normalized format. Figure 3 presents a high-level approach on the entire process.

6.1 *Step 1 – VAE training*

In the first step, we utilized the time-series for the first 200 units to train the VAE. We used most of the units in the first step to decrease the decoder bias, since it is an important element for accurately predicting RUL estimates. Hence, we ensure that most of the aleatoric variability on the training dataset is captured and the resulting decoder can be considered as an adequate substitute of the CMAPSS simulator, and as extension, the RL environment.

The encoder and decoder models were designed using fully connected deep neural networks, with layer sizes $l_{VAE} = (256, 128, 64)$ and rectified linear units (reLU) for activation functions on each layer. The sampling layer was designed to follow a two-dimensional diagonal Gaussian distribution. Training ran for 30 epochs.

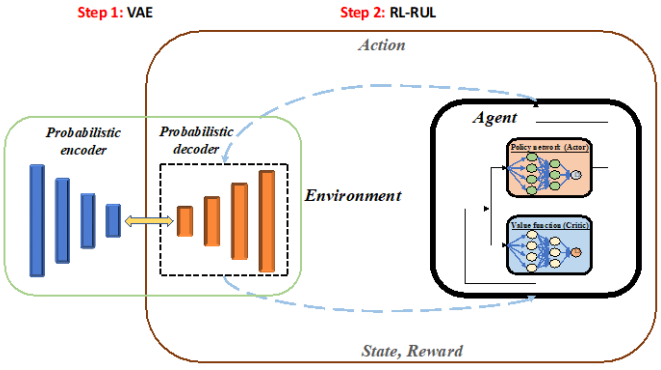

Figure 3: Illustration of the proposed method

6.2 *Step 2 – RL agent training*

In the second step, the trained decoder formed the environment for the RL agent training. A subset of 40 units was selected in that step. For parameterized policy, we used a fully connected deep neural network with layer weights $l_\pi = (512, 256, 128, 64)$ and reLU activation functions for each layer. Also, since the policy was stochastic, it returned actions and their log standard deviations. A total number of 300 episodes was performed to ensure stable convergence of the agent training.

6.3 *Results*

Below, we show RUL results for 6 randomly selected engines from the testing set that start at random time instances using the trained RL agent (Figure 4). Further, in Table 1 we provide average values of true RULs and their estimated counterparts (as a % of actual lifetime) for the last 20 time cycles of 100 random sampled units from the testing set (with replacement).

We see that the agent manages to approximate the true RUL values, especially when the estimation occurs relatively late in the unit's lifetime. There are also cases where the results show a small divergence, either above or below true values, which is more common when we perform RUL estimation early in the system's lifetime. That is understandable, since there is a significant remaining life uncertainty for any system that is early on its service, and the RL agent cannot capture it. Nevertheless, as is the case in most real-world industrial systems, RUL estimation cycles are performed regularly, and with more sensor observations collected during that time, the RL agent will provide more confident results.

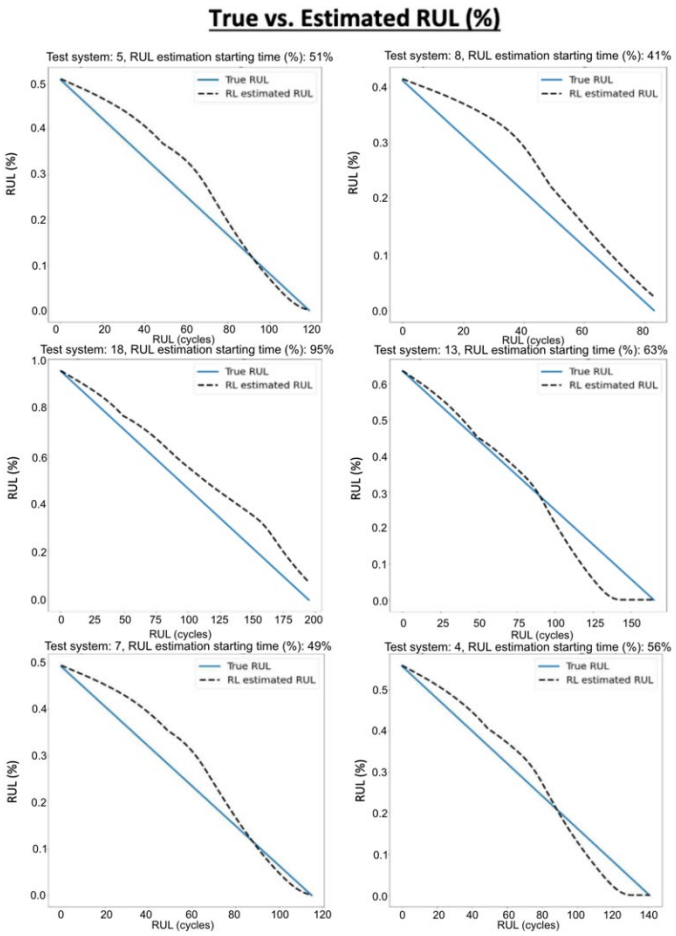

Figure 4: True vs. Estimated RUL results for randomly sampled units in the test set.

To further establish the validity of our method, we compared root mean squared error (RMSE) values between true

and estimated RUL for all twenty engines on the testing set against those obtained using three well-established predictive maintenance methods: i) the Cox proportional hazards model with time-varying coefficients, ii) linear regression, and iii) a two-layered LSTM deep neural network with a lookback period of 20 timesteps (Table 2). The proposed method provides the best results while utilizing minimal data preprocessing.

Table 1: Average True vs. Estimated RUL values (%)

Time cycle	True RUL (%)	Estimated RUL (%)	Time cycle	True RUL (%)	Estimated RUL (%)
19	8.27	8.99	**9**	3.93	6.17
18	7.84	8.69	**8**	3.49	5.93
17	7.4	8.39	**7**	3.1	5.69
16	6.97	8.09	**6**	2.65	5.45
15	6.53	7.8	**5**	2.21	5.17
14	6.1	7.52	**4**	1.76	4.9
13	5.66	7.25	**3**	1.32	4.69
12	5.24	6.96	**2**	0.88	4.48
11	4.8	6.69	**1**	0.44	4.28
10	4.36	6.43	**0**	0	4.08

Table 2: RMSE value comparison

	Our method	*Cox PH with time-varying coefficients*	*LSTM*	*Linear regression*
RMSE	**29.89**	39.34	55.16	60.11

7. DISCUSSION & FUTURE WORK

The proposed method provides significant advantages compared to traditional filtering and predictive maintenance approaches. First, it does not require predefined state/observation equations that can be poor approximations of the true latent state evolution and state mapping to observations. Second, it can estimate RUL values for previously unseen time-series data and at the same time provides a low-cost, relatively high-fidelity surrogate model for the actual system under observation. Also, the agent manages to approximate latent system dynamics using a model-free approach, thus making it highly modular; the proposed method can be easily modified by substituting the decoder with any off-the-shelf simulator, without the need to overhaul the RL step. Usually, predictive maintenance methods require extensive data preprocessing such as introducing features for breakdown, observation start and stop, careful feature selection based on goodness of fit and coefficient *p*-values, hyperparameter tuning, upper clipping for stability. None of these were necessary for our method, thus greatly simplifying the overall process of RUL estimation.

Despite its appealing properties in RUL estimation, there are ways that the proposed method can be further improved upon, such as more robust surrogate environment models that can better capture time dependencies between successive latent states, including multiple levels of latent processes. Furthermore, the proposed method in its current form is hardcoded to take inputs in a tabular format. However, we plan to expand its capabilities by making it possible to accept high-dimensional data (e.g., images) as inputs, thus avoiding many of the data preprocessing assumptions performed here, leading to a more realistic prognostics model.

In the future, we plan to address the issues stated above. We also want to test the method on publicly available real-world degradation datasets, where the systems do not exhibit monotonic degradation signals as is the case with CMAPSS. Finally, we also want to investigate safe RL methods, which are crucial to contemporary industrial and manufacturing settings.

ACKNOWLEDEGEMENT

This material is based upon work supported by the National Science Foundation under Grant No. 1846975.

REFERENCES

1. Saxena, A., Goebel, K., Simon, D., and Eklund, N., "Damage propagation modeling for aircraft engine run-to-failure simulation." *2008 IEEE International Conference on Prognostics and Health Management*, 2008, pp. 1-9.
2. Wang, Qi. "VARL: a variational autoencoder-based reinforcement learning Framework for vehicle routing problems." *Applied Intelligence 52(8),* 2022, pp. 8910–8923.
3. Hu, Y., Sun S. "RL-VAEGAN: Adversarial defense for reinforcement learning agents via style transfer." *Knowledge-Based Systems 221*, 2021: 106967.
4. Baucum, M., Khojandi, A., and Vasudevan, R., "Improving deep reinforcement learning with transitional variational autoencoders: A healthcare application." *IEEE Journal of Biomedical and Health Informatics 25(6)*, 2020. pp. 2273-2280.
5. Yarats, D., Zhang, A., Kostrikov, I., Amos, B., Pineau, J., Fergus, R., "Improving sample efficiency in model-free reinforcement learning from images." *Proceedings of the AAAI Conference on Artificial Intelligence 35(12),* 2021, pp. 10674-10681.
6. Kim, H., Masanori, Y., Miyoshi, K., Iwata, T., Yamakawa, H., "Reinforcement Learning in Latent Action Sequence Space.", *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2020, pp. 5497-5503.
7. Andersen, P.A., Goodwin, M., Granmo, O.C., "Towards safe reinforcement learning in industrial grid-warehousing", *Information Sciences 537*, 2020, pp. 467-484.
8. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 'Proximal Policy Optimization algorithms", *arXiv preprint arXiv:1707.06347*, 2017.

BIOGRAPHIES

Erotokritos Skordilis, PhD,
Department of Business Technology
Miami Herbert Business School,
University of Miami,
5250 University Drive
Coral Gables, FL, USA

Erotokritos Skordilis received the Ph.D. degree from the University of Miami in 2019. He worked as a post-doctoral researcher with the National Renewable Energy Laboratory's Computational Science Center, and currently is a lecturer at the University of Miami, Herbert Business School. His research focuses on scalable reinforcement learning applications.

Ramin Moghaddass, PhD,
Industrial & Systems Engineering,
University of Miami,
1251 Memorial Drive
Coral Gables, FL, USA

Ramin Moghaddass is an Associate Professor in the College of Engineering's Department of Industrial & Systems Engineering and the Director of the DOE Industrial Assessment Center and Data Analytics Lab. Dr. Moghaddass studies complex, sensor-driven engineered systems. His research is on developing a new generation of flexible and large-scale models for real-time system health monitoring inspired by dynamic structures and networks.

Md Tanzin Farhat, PhD student
Industrial & Systems Engineering,
University of Miami,
1251 Memorial Drive
Coral Gables, FL, USA

Md Tanzin Farhat received his MS in computer engineering from University of Toledo in 2018. He is currently a PhD student working on modeling degradation systems with machine learning models for prognostics, reliability and system health monitoring.